show toc

Organizing Software Development by Project Using Microsoft Visual SourceSafe

Microsoft Corporation

May 1995

Abstract

Professional software development requires a streamlined process for managing changes to the source code for each application under development. Microsoft® Visual SourceSafe™ records the change history at the system, project, and file levels, allowing you to safely distribute development among multiple programmers, track changes, and recover old versions of individual files or of an entire application.

Introduction

Code is a precious resource. To protect it, most developers safeguard files against unauthorized changes and unexpected failures using some system of version control. These systems can vary from gentleman's agreements about commenting changes and storing old versions to sophisticated software systems that track changes and histories automatically.

Most of these source control systems work well for individual source files. However, almost all of them *fail to establish relationships between files*. This can pose quite a problem in the Microsoft® Windows® environment, where a single application can consist of multiple executable files and dynamic link libraries built out of many different source files, which, in turn, may be reused among many other applications. Today, it is as important to manage the relationships between source files as it is to protect the contents of the source files themselves.

Microsoft Visual SourceSafe™ version control software solves this problem by combining the tasks of project management and source control. By focusing on projects as well as source files, Visual SourceSafe provides an elegant solution to problems not easily solved using standard, file-oriented source control systems.

Streamlining Software Development

To understand the benefits of project-oriented source control, simply compare it to a file-oriented system. A standard version control system (such as the UNIX utility RCS) is essentially a collection of tools that operate on individual files, controlling file access and updates and comparing previous versions. To operate on a group of files, you need to write a batch file or specify wildcards on the command line.

Microsoft Visual SourceSafe stores files in a central database on the network, rather than on an ordinary DOS directory. At the system level, this database appears as a "black box." When viewed through Visual SourceSafe, however, the database is seen to contain all your source files and histories organized into project hierarchies.

When you retrieve a file, Visual SourceSafe marks the file as checked out to you in its database and then allows you to make changes to the file on your machine. When you check the file back in, Visual SourceSafe updates its database and changes the file's access privilege on your machine back to read-only. So how is this any different from file-oriented source control?

With each change, the Visual SourceSafe database records and tracks project information not available to file-oriented systems. Each time a file is added, modified, shared, moved, or deleted from a project, Visual SourceSafe updates both the file *and* the project's history. You can use the project history to simplify these tasks:

- Viewing the status of all the files in a specific project and all subordinate projects prior to building.
- Zooming in on a specific file change that might have caused a bug in a build on a certain date.
- Recreating any previous version of an entire application.
- Maintaining source files that are shared among many different applications.
- Determining which projects are affected by changing a file that is shared among many different applications.
- Managing client-specific versions of a general application.

Attempting these tasks with file-oriented systems can be an incredible chore and a frustrating impediment to software developers. Visual SourceSafe project-oriented version control streamlines the development process by making all of these tasks straightforward, as illustrated by the following scenarios.

Preparing for a Build

Suppose that you are about to build a major application consisting of many separate components. Before you start the build, you want to make sure that no one is revising code at the last minute, or, in version control terms, that none of the files in the entire system is checked out.

A standard version control system provides you with a tool for determining whether a file is checked out. Your job is to run this utility on every file in every directory that is being used for the build. Batch files and wildcards make the task easier, but in a complicated system, it can still be quite a chore.

Visual SourceSafe, like other systems, can determine whether a file is checked out. But it can also create a higher-level report: a list of all the checked-out files in a project. This feature becomes even more powerful when used recursively to include all subprojects in the current project. Visual SourceSafe checks every file in every relevant project and generates a list of every checked-out file. You know immediately whether you can proceed with the build (or who to talk to if you can't!). By simply executing a command on a project, Visual SourceSafe automates a previously tedious, manual task.

Pinpointing Regressions

File history reports are available in all version control systems, including Visual SourceSafe. File history lists each version of a file, from the most recent to the oldest, with information such as what happened to the file, who did it, when it was done, and what comment was made.

Despite their usefulness, file histories have severe limitations. For instance, suppose a feature that was working correctly last week breaks in this week's build of your application. Obviously, someone introduced this bug recently, but in what file?

To tackle this with a standard version control system, you would generate a history report on a likely-looking file, see whether it had changed recently, and look through the changes. If you don't find the bug, you pick another file to check, and so on. You might go through every file in the system this way and never find the critical change—because the change was actually adding or deleting a file, which standard version control systems don't track at all!

With Visual SourceSafe, you generate a report on the *project itself*. For instance, it might report that COMMON.BAS was just modified; before that, OPENALL.FRM was changed; before that, FILESUPP.BAS was added to the project; and so on. Visual SourceSafe collates the changes that you would otherwise have to sort through manually, enabling you to view the order of changes over the last week. This can save you a lot of time and help you avoid dead ends.

Recreating Previous Project Versions

By tracking project history, Visual SourceSafe allows you to quickly recreate previous versions of an entire application. This can help you resolve bugs reported from previous shipped versions to make sure they were fixed in the version currently under development.

For instance, suppose a client reports a printing problem in version 2.03 of an application. That version of the application may have included version 10 of one file, version 15 of another, and so on; but you don't need to worry about that. By requesting the specific version of the project from Visual SourceSafe, you can restore a complete, local copy of the application's source files used to build version 2.03.

To do this with a standard version control system, you either have to separately archive the sources of each released version of the application or track the specific file versions for each release. Either way, restoring the correct source files for a previous build becomes a tedious, manual process—a task that is likely to be put off or delayed.

Maintaining Reusable Code

Most applications are developed around a common base of *core code*. These files are used over and over again in many different applications and usually evolve over time, receiving bug fixes, performance improvements, and new features. The benefits of reusing existing code are enormous, but so are the headaches when managing the organizational issues. You have to remember which applications are using each file and propagate every change to all the appropriate places. This is only a minor annoyance when five applications are reusing one file; but when twenty applications are mixing and matching fifty different reusable files, you've got a problem!

A standard version control system can't help at all with this problem because one file can exist simultaneously in many different projects. But Visual SourceSafe can automate it completely. Within its database, Visual SourceSafe stores each file only once. Each project that contains a file has a pointer to the location of the file in the database. All versions of the file are available to each project, and a project may "freeze" the version of a file to avoid introducing errors while another development team works on the reused code.

To take a common example, suppose that you have a single source file that contains many different procedures for printing reports. In Visual SourceSafe, each application that needs to print reports would share the file. If you find a bug, you update the file from any project—the change is instantly propagated to every project that shares the file. Then Visual SourceSafe can report on which projects share the file, so you know which applications are affected and may need to be rebuilt.

Creating Client-Specific Versions

Another common source control problem concerns clients who want applications customized to meet their specific needs. Essentially, you have many different applications that share almost *all* of the same source files. Using standard source control tools, tracking changes and keeping builds straight can take more time than the programming. Using Visual SourceSafe, you create a project for each new client, indicating which files are shared and which files are unique. When you work on a project, client-specific changes stay with the current project and changes to shared files are propagated to all client versions.

Send feedback on this article. Find support options.

HOWTO: Write Automation for Visual SourceSafe 5.0/6.0

Article ID: Q201431

Last Reviewed: February 4, 2000

Send to a friend

Provided by Microsoft Product Suppo Services

The information in this article applies to:

- Microsoft Visual SourceSafe, 32-bit, for Windows versions 5.0, 6.0
- Microsoft Visual C++, 32-bit Editions, version 6.0

SUMMARY

This article provides sample C++ code for OLE Automation calls. It also describes how to check whether version 5.0 or 6.0 of Visual SourceSafe is running on a computer because OLE Automation written for one version may not run under the other.

MORE INFORMATION

Visual SourceSafe must be registered before you can use its OLE Automation Model. The CLSID {783CD4E4-9D54-11CF-B8EE-00608CC9A71F} is located in HKEY_CLASSES_ROOT/CLSID/. Registration occurs by default when you install the Visual SourceSafe Client.

Although it uses a different method, the following Microsoft Knowledge Base article describes how to establish a connection to a Visual SourceSafe database:

Q169928 HOWTO: Open a SourceSafe Database with OLE Automation in C++

Another method is to use the #import call to connect to the Ssapi.dll file. If working with both 5.0 and 6.0 clients, you must import this file for both versions. The easiest way to do this is to import one Ssapi.dll into another namespace that can be called later. Then use the code found under Sample Code, below, to find out which version of Visual SourceSafe the computer has and call the correct functions.

NOTE: The development computer must be able to access both the 6.0 and 5.0 Ssapi.dll. You can load them as follows:

```
// This path needs to point to the ssapi for the 6.0 version of
// SourceSafe (version 8169)
#import "C:\Visual Studio\Vss\Win32\ssapi.dll"



// This path needs to point to the ssapi for the 5.0 version of
// SourceSafe (version 2218-2222)
#import "C:\DevStudio\Vss\Win32\ssapi.dll" rename_namespace("vss5")
```

Then, you can just test them with:

```
// version was initialized already by calling -
// version = checker.checkVersion(); where checker is class Check
if (version == 5) {
   using namespace vss5;
   .... // all your SourceSafe 5 commands go here.
} else {
   .... // all your SourceSafe 6 commands go here.
}
```

Sample Code

The following sample code demonstrates how to check which version of SourceSafe a computer is running. It creates a class called CCheck that has the function that does the checking.

```
// CCheck::checkVersion returns:
//    -1 on error
//     5 for VSS version 5.0
//     6 for VSS versions later than 5.0
#include <windows.h>

class CCheck
{
public:

    int checkVersion();
private:
    LPTSTR GetRegistryInfo();
    TCHAR m_szVSSDir[MAX_PATH];
};

LPTSTR CCheck::GetRegistryInfo()
{
   m_szVSSDir[0] = '\0';

   // Find the ssapi.dll registry key.
   HKEY hclass;
   if (RegOpenKeyEx(HKEY_CLASSES_ROOT,
                    "CLSID",
                    0,
                    KEY_QUERY_VALUE,
                    &hclass) == ERROR_SUCCESS) {
      HKEY hkey;
      if (RegOpenKeyEx(hclass,
                       "{783CD4E4-9D54-11CF-B8EE-00608CC9A71F}",
                       0,
                       KEY_QUERY_VALUE,
                       &hkey) == ERROR_SUCCESS) {
         HKEY hproc;
         if (RegOpenKeyEx(hkey,
                          "InprocServer32",
                          0,
                          KEY_QUERY_VALUE,
                          &hproc) == ERROR_SUCCESS) {
            DWORD dwType = 0;
            DWORD dwSize = sizeof(m_szVSSDir);
            RegQueryValueEx(hproc,
                            "",
                            NULL,
                            &dwType,
                            (LPBYTE)m_szVSSDir,
                            &dwSize);
            RegCloseKey(hproc);
         }
         RegCloseKey(hkey);
      }
      RegCloseKey(hclass);
   }
   return m_szVSSDir;
}

int CCheck::checkVersion()
{
   LPTSTR szPath;
   DWORD dwVerHnd;
```

```
        DWORD dwVerInfoSize;
        char szFullPath[MAX_PATH+1];
        long resMOld = 327680;
        long resM;
        int ret = -1;

        szPath = GetRegistryInfo();

        strcpy(szFullPath, szPath);
        dwVerInfoSize = GetFileVersionInfoSize(szFullPath, &dwVerHnd);
        if (dwVerInfoSize) {
           // If we were able to get the information, process it:
           HANDLE hMem;
           LPVOID lpvMem;
           BOOL fRet;
           UINT cchVer = 0;
           VS_FIXEDFILEINFO *vInfo;

           hMem = GlobalAlloc(GMEM_MOVEABLE, dwVerInfoSize);
           lpvMem = GlobalLock(hMem);
           GetFileVersionInfo(szFullPath, dwVerHnd, dwVerInfoSize, lpvMem);

           fRet = VerQueryValue(lpvMem, TEXT("\\"),
                                (LPVOID*)&vInfo, &cchVer);
           if (fRet && cchVer && vInfo) {
               resM = vInfo->dwFileVersionMS;
               if (resMOld >= resM) ret = 5;
               else ret = 6;
           }
           GlobalUnlock(hMem);
           GlobalFree(hMem);
        }
        return ret;
     }
```

REFERENCES

MSDN Library: Visual SourceSafe OLE Automation

For additional information, please see the following articles in the Microsoft Knowledge Base:

> Q169928 HOWTO: Open a SourceSafe Database with OLE Automation in C++

Also see the article "Visual SourceSafe 6.0 Automation" on the Microsoft Developer Network Web site:

> http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/techart/vssauto.htm

Additional query words: kbDSupport

Keywords : kbAutomation kbSSafe500 kbSSafe600 kbVC600
Issue type : kbhowto
Technology : kbvc

show toc

Visual SourceSafe 6.0 Automation

Tim Winter
Microsoft Corporation

September 1998

Click to copy the files associated with this technical article.

Summary: Discusses the Automation interface exposed by Microsoft® Visual SourceSafe™ version 6.0. Written from the perspective of the Microsoft Visual Basic® programmer and includes code snippets written in Visual Basic version 5.0. (67 printed pages)

Contents

Introduction

This article discusses the Automation interface exposed by Visual SourceSafe version 6.0. It is written from the perspective of the Microsoft Visual Basic® programmer and includes code snippets written in Visual Basic version 5.0. In addition, there are several Automation code samples available with this article. Future versions of Visual SourceSafe will attempt to make as few changes as possible to this interface. Earlier (pre-5.0) versions of Visual SourceSafe do not support this interface in any way.

This article is divided into two parts: "Trapping SourceSafe Events" and "Driving a SourceSafe Database". SourceSafe's Automation interface allows the user to hook into and control certain events as they occur within a SourceSafe session. For example, you may want to create an application that would notify specific users via e-mail whenever certain files have been checked in or checked out from the database. Trapping SourceSafe events allows you to do this. The second part of this article describes the objects, methods, and properties required to create your own interface in the SourceSafe database.

Terms and Expressions

The following terms and expressions are used throughout this article:

- <...>: Refers to a section of code not included in the code sample. This could represent a section of code required for the sample to execute properly (declarations, definitions, and so on).
- boolVar: Refers to a Boolean variable.
- objVSSDatabase: Refers to a SourceSafe database object created with the Open method of the VSSDatabase object. (See the "Open Method" topic in the "VSSDatabase Object" section for more information).
- objVSSItem: Refers to a created file or project object.
- strVar: Refers to a string variable.

Trapping SourceSafe Events

IVSSEventHandler Interface

The VSSApp object allows the user to hook into and control certain events as they occur within a SourceSafe session. For example, you may want to create an application that would notify specific users via e-mail whenever certain files have been checked in or checked out from the database. Trapping SourceSafe events allows you to do this.

From Visual Basic, this may be accomplished through the creation of a Visual SourceSafe add-in (which is typically a Microsoft ActiveX® DLL). When SourceSafe is launched, it will create an instance of your add-in and call its Init method. This method is exposed through the IVSSEventHandler interface.

To trap SourceSafe events, your ActiveX DLL must support the IVSSEventHandler interface. In Visual Basic, this is best accomplished using the IMPLEMENTS statement. Once you have set a reference to the SourceSafe 6.0 Type Library, you can implement the IVSSEventHandler interface with the code:

```
Implements IVSSEventHandler
```

When Visual Basic implements the interface, it provides its own version of every procedure exposed by the interface. Your add-in must include code for each of these procedures, regardless of whether or not you use them. For more information on these procedures, see the "VSSApp Object" section of this article.

Methods

Init(pIVSS As VSSApp)

The IVSSEventHandler interface exposes the Init method, which is called by SourceSafe when it creates an instance of your add-in (this takes place immediately after login). Because SourceSafe calls this method, any add-in you create must support it. The Init method accepts a single parameter: *pIVSS*. This required parameter represents the IVSS interface of the VSSApp object. To the Visual Basic programmer, this interface will appear as the VSSApp object itself.

When SourceSafe calls the Init method of your add-in, it will pass the VSSApp object to the add-in automatically. This object allows your add-in to control SourceSafe.

The following Visual Basic code could be placed in the Init method of your add-in to display login information on the current SourceSafe session:

```
Sub IVSSEventHandler_Init(ByVal pIVSS As VSSApp)

   Set VSSHandler = pIVSS
   MsgBox ("User " + pIVSS.VSSDatabase.Username + _
   " has just logged into the database located at " + _
   pIVSS.VSSDatabase.SrcSafeIni)

End Sub
```

For more information on creating a SourceSafe add-in and hooking into SourceSafe events, see the "VSSApp Object" and the "Putting It All Together" sections of this article.

VSSApp Object

The VSSApp object allows the user to hook into and control certain events as they occur within a SourceSafe session. The VSSApp object exposes two interfaces: IVSS and IVSSEvents. The IVSS interface provides a VSSDatabase property while the IVSSEvents interface provides the events listed here.

BeforeAdd	AfterAdd
BeforeBranch	AfterBranch
BeforeCheckIn	AfterCheckIn
BeforeCheckOut	AfterCheckOut
BeforeEvent	AfterEvent
BeforeRename	AfterRename
BeforeUndoCheckOut	AfterUndoCheckOut
BeginCommand	EndCommand

To the Visual Basic programmer, these interfaces appear to come directly off the VSSApp object. For example, if you wanted to know the UserName of a person logging into SourceSafe, the following Visual Basic code would provide that information:

```
Implements IVSSEventHandler
' The next statement creates an instance of a VSSApp Object
Dim WithEvents VSSHandler As VSSApp
Sub IVSSEventHandler_Init(ByVal pIVSS As VSSApp)
   Set VSSHandler = pIVSS
   MsgBox ("User " + pIVSS.VSSDatabase.Username + _
   " has just logged into SourceSafe.")
End Sub
```

The preceding example uses a third interface called IVSSEventHandler that exposes an Init method. (For more information, see the "IVSSEventHandler Interface" section earlier in this article.) The Init method is called by SourceSafe when it creates an instance of your add-in, so your add-in must support it.

We recommend you make the VSSApp object global. This allows you to gain additional information on the database object from within each event.

Trapping SourceSafe Events—An Overview

When the SourceSafe Explorer or Admin utility is opened, it looks for a file called ssaddin.ini. This file contains information on the SourceSafe add-ins that you want to launch for that SourceSafe session. If this file exists, SourceSafe attempts to create an instance of any add-ins referenced within the file.

The following steps describe how to set up your SourceSafe add-in from Visual Basic:

1. Create and compile an ActiveX DLL in Visual Basic that implements the IVSSEventHandler interface (this is described in detail later).
2. Open the Windows registry, and locate the ProgID of your DLL.
3. Create (or open) the text file ssaddin.ini, and add a reference to the ProgID of your DLL. This reference must use the syntax:

   <ProgID>=1

The file ssaddin.ini must reside in the same folder as ssapi.dll. By default, this is the ...\VSS\WIN32 folder.

Note To trap events in SourceSafe, each SourceSafe installation must have its own copy of ssaddin.ini and the ActiveX DLL (or add-in). For example, suppose you have SourceSafe installed on Server A and a client installation of SourceSafe on workstations B and C. The file ssaddin.ini must appear in the ...\VSS\WIN32 folder of A, B, and C, and you must install and register the add-in on each system (A, B, and C).

If you were to add a third SourceSafe installation to the preceding example (workstation D) and not add both of these files, SourceSafe events that occur on workstation D will not be trapped (although events on A, B, and C will continue to be trapped).

Launch SourceSafe. SourceSafe will create an instance of your add-in and call its Init method. The Init method of your add-in is passed a VSSApp object, which establishes the communication link between it and SourceSafe.

To trap SourceSafe events, your ActiveX DLL must support the IVSSEventHandler interface. In Visual Basic, this is best accomplished using the IMPLEMENTS statement. Once you have set a reference to the SourceSafe 6.0 Type Library, you can implement the IVSSEventHandler interface with the code:

```
Implements IVSSEventHandler
' The next statement creates an instance of a VSSApp Object
Dim WithEvents VSSHandler As VSSApp
```

When Visual Basic implements the interface, it provides its own version of every procedure exposed by the interface. Your add-in must include code for each of these procedures regardless of whether or not you use them. For example, if your add-in only deals with the CheckOut and CheckIn events, you must still add empty procedures (procedures that don't do anything) for every other event that the interface exposes. These procedures (or events) are documented in the next section.

Events

All of the following SourceSafe events may be trapped from your add-in. The sample code used in this section assumes your add-in was created using an ActiveX DLL.

Each of these events is passed a *VSSItem* parameter that exposes many properties and methods accessible from your add-in. For more information on the VSSItem object, see the "VSSItem Object" section later in this article.

Each of the Before*<event>* functions (BeforeAdd, BeforeCheckIn, and so on) allows you to prevent the event from occurring. This is accomplished by setting the function's return value to false.

Note Each of these events is only fired on the client machine where the SourceSafe action is taking place. In other words, if a user is adding a file on machine A, the BeforeAdd event is fired only on machine A.

AfterAdd(pIItem As VSSItem, Local As String, Comment As String)

The AfterAdd event is fired immediately after an item is added to the SourceSafe database. This event accepts three parameters that are passed to your add-in from Visual SourceSafe:

- **pIItem:** This *VSSItem* parameter accepts a **VSSItem** object representing the file added to SourceSafe.
- **Local:** This string parameter accepts the complete path of the file added to SourceSafe.
- **Comment:** This string parameter accepts the comment (if any) applied by the user when the file was added.

The following Visual Basic code displays a message box with the name of the file that was just added, the project path it was added to, and the comment applied (if any):

```
Sub VSSHandler_AfterAdd(ByVal Item As IVSSItem, ByVal LocalSpec _
As String, ByVal Comment As String)
MsgBox ("The file " + LocalSpec + " was just added to " + _
Item.Parent.Spec _
   + " with a comment of " + Comment)
End Sub
```

This event is fired after adding file objects only; it is not fired when a new project object is created.

AfterBranch(pIItem As VSSItem, Comment As String)

The AfterBranch event is fired immediately after an item is branched in the SourceSafe database. This event accepts two parameters that are passed to your add-in from Visual SourceSafe:

- **pIItem:** This *VSSItem* parameter accepts a **VSSItem** object representing the file branched in SourceSafe.
- **Comment:** This string parameter accepts the comment applied by the user when the file was branched.

The following Visual Basic code displays the name of the branched file, the name of the project from which it was branched, and the comment applied to the branch by the user (if any):

```
Sub VSSHandler_AfterBranch(ByVal Item As IVSSItem, ByVal Comment _
As String)
   MsgBox ("The file " + Item.Name + " in the project " + _
   Item.Parent.Name + " was just branched with a comment of " + _
   Comment)
End Sub
```

The AfterBranch event is only fired when file objects are branched. SourceSafe does not support branching project objects at this time.

AfterCheckIn(pIItem As VSSItem, Local As String, Comment As String)

The AfterCheckIn event is fired immediately after an item is checked into the SourceSafe database. This event accepts three parameters that are passed to your add-in from Visual SourceSafe:

- **pIItem:** This *VSSItem* parameter accepts a **VSSItem** object representing the file checked in to SourceSafe.
- **Local:** This string parameter accepts the complete path of the file checked in to SourceSafe.
- **Comment:** This string parameter accepts the comment applied by the user when the file was checked in.

This event is fired after checking in a file or project object. The following Visual Basic code displays a message box with the name of the file that was just checked in, the local path it was checked in from, and the comment applied (if any):

```
Sub VSSHandler_AfterCheckIn(ByVal Item As IVSSItem, ByVal LocalSpec _
As String, ByVal Comment As String)
   MsgBox ("The file " + LocalSpec + " was just checked in to " + _
   Item.Parent.Spec + " with a comment of " + Comment)
End Sub
```

When you check in a project object, each file that is checked in to the project will fire this event.

Note A shared file may be checked in from any one of several locations. The links property of the IVSSItem can be used to retrieve a list of these locations.

AfterCheckOut(pIItem As VSSItem, Local As String, Comment As String)

The AfterCheckOut event is fired immediately after an item is checked out from the SourceSafe database. This event accepts three parameters that are passed to your add-in from Visual SourceSafe:

- **pIItem:** This *VSSItem* parameter accepts a **VSSItem** object representing the file checked out from SourceSafe.
- **Local:** This string parameter accepts the complete path of the file checked out to SourceSafe.
- **Comment:** This string parameter accepts the comment applied by the user when the file was checked out.

This event is fired after checking out a file or project object. The following Visual Basic code displays a message box with the name of the file that was just checked out, the local path it was checked out to, and the comment applied (if any):

```
Sub VSSHandler_AfterCheckOut(ByVal Item As IVSSItem, ByVal _
LocalSpec As String, ByVal Comment As String)
   MsgBox ("The file " + LocalSpec + " was just checked out to " + _
   Item.Parent.Spec + " with a comment of " + Comment)
End Sub
```

When you check out a project object, each file that is checked out from the project will fire this event.

AfterEvent(iEvent As Long, pIItem As VSSItem, Str As String, var)

Currently this event serves no purpose. It may be used in future versions of SourceSafe.

AfterRename(pIItem As VSSItem, OldName As String)

The AfterRename event is fired immediately after a file or project item is renamed in the SourceSafe database. This event accepts two parameters that are passed to your add-in from Visual SourceSafe:

pIItem: This *VSSItem* parameter accepts a VSSItem object representing the file or project that was renamed in the SourceSafe database.

OldName: This string parameter accepts the original name of the file or project that was renamed in the SourceSafe database.

The following Visual Basic code displays a message box that displays the old and new name of the renamed file or project object:

```
Sub VSSHandler_AfterRename(ByVal Item As IVSSItem, ByVal OldName _
As String)
   Dim strItemType As String
   If Item.Type = VSSITEM_FILE Then
      strItemType = "file"
   Else
      strItemType = "project"
   End If
   MsgBox ("The " + strItemType + " " + OldName +  _
   " was just renamed to " + Item.Name)
End Sub
```

AfterUndoCheckOut(pIItem As VSSItem, Local As String)

The AfterUndoCheckOut event is fired immediately after an item is unchecked out from the SourceSafe database. This event accepts two parameters that are passed to your add-in from Visual SourceSafe:

- **pIItem:** This *VSSItem* parameter accepts a **VSSItem** object representing the file that was unchecked out from SourceSafe.
- **Local:** This string parameter accepts the complete path of the file that was unchecked out from SourceSafe.

The following Visual Basic code displays a message box describing the SourceSafe file path that was unchecked out and the local folder from which it was unchecked out.

```
Sub VSSHandler_AfterUndoCheckOut(ByVal Item As IVSSItem, _
ByVal LocalSpec As String)
   MsgBox ("The file " + Item.Spec + " was just unchecked out from " + _
   "the folder " + LocalSpec)
End Sub
```

When you uncheck out a project object, each file that is unchecked out to the project will fire this event.

BeforeAdd(pIPrj As VSSItem, Local As String, Comment As String)

The BeforeAdd event is fired immediately before an item is added to the SourceSafe database. This event accepts three parameters that are passed to your add-in from Visual SourceSafe:

- **pIItem:** This *VSSItem* parameter accepts a **VSSItem** object representing the file being added to SourceSafe.
- **Local:** This string parameter accepts the complete path of the file being added to SourceSafe.
- **Comment:** This string parameter accepts the comment applied by the user to the file being added.

This event is fired prior to adding a file only (adding a project will not fire this event). The following Visual Basic code displays a message box with the name of the file that will be added, the project path it will be added to, and the comment being applied. If there is no comment, the add is canceled:

```
Function VSSHandler_BeforeAdd(ByVal Prj As IVSSItem, _
ByVal LocalSpec As String, ByVal Comment As String) As Boolean
   MsgBox ("The file " + LocalSpec + " is about to be added to
              " +  Prj.Spec _+ " with a comment of " + Comment)
   If Comment = "" Then
```

```
        VSSHandler_BeforeAdd = False
        Else
        VSSHandler_BeforeAdd = True
        EndIf
End Function
```

If you set the function VSSHandler_BeforeAdd to false, the addition of the file object will not occur.

This event does not support adding files from the folder level. For example, if the user attempts to add the contents of the folder "C:\MyFolder," the error "Cannot use wildcards with this command" will be generated, and no items will be added.

When you add multiple files from a common folder, each file being added will fire this event.

BeforeBranch(pIItem As VSSItem, Comment As String)

The BeforeBranch event is fired prior to branching a file item in the SourceSafe database. This event accepts two parameters that are passed to your add-in from Visual SourceSafe:

- **pIItem:** This *VSSItem* parameter accepts a **VSSItem** object representing the file branched in SourceSafe.
- **Comment:** This string parameter accepts the comment applied by the user when the file was branched.

The following Visual Basic code displays the name of the file to be branched, the name of the project from which it will be branched, and the comment being applied to the branch by the user. If no comment has been added, the branch is canceled:

```
Function VSSHandler_BeforeBranch(ByVal Item As IVSSItem, _
ByVal Comment As String) As Boolean
   MsgBox ("The file " + Item.Name + " in the project " + _
       Item.Spec + " is about to be branched with a comment of " + _
       Comment)
   If Comment = "" Then
      VSSHandler_ BeforeBranch = False
      Else
      VSSHandler_ BeforeBranch = True
   EndIf
End Function
```

If you set the function VSSHandler_BeforeBranch to false, the branching of the file object will not occur.

The BeforeBranch event is only fired when file objects are branched. SourceSafe does not support branching project objects at this time.

BeforeCheckIn(pIItem As VSSItem, Local As String, Comment As String)

The BeforeCheckIn event is fired immediately prior to the check in of a file object to the SourceSafe database. This event accepts three parameters that are passed to your add-in from Visual SourceSafe:

- **pIItem:** This *VSSItem* parameter accepts a **VSSItem** object representing the file to be checked in to SourceSafe.

- **Local:** This string parameter accepts the complete path of the file to be checked in to SourceSafe.
- **Comment:** This string parameter accepts the comment applied by the user of the file to be checked in to SourceSafe

The following Visual Basic code displays a message box with the name of the file that is to be checked in, the local path it will checked in from, and the comment applied. If no comment has been added, the check-in is canceled:

```
Function VSSHandler_BeforeCheckIn(ByVal Item As IVSSItem, _
ByVal LocalSpec As String, ByVal Comment As String) As Boolean
   MsgBox ("The file " + LocalSpec + " is about to be checked in to " + _
   Item.Parent.Spec + " with a comment of " + Comment)
   If Comment = "" Then
      VSSHandler_ BeforeCheckIn = False
      Else
      VSSHandler_ BeforeCheckIn = True
   EndIf
End Function
```

If you set the function VSSHandler_BeforeCheckIin to false, the checking in of the file object will not occur.

When you check in a project object, each file that is checked in to the project will fire this event.

BeforeCheckOut(pIItem As VSSItem, Local As String, Comment As String)

The BeforeCheckOut event is fired immediately prior to the check out of a file item from the SourceSafe database. This event accepts three parameters that are passed to your add-in from Visual SourceSafe:

- pIItem: This VSSItem parameter accepts a VSSItem object representing the file being checked out from SourceSafe.
- Local: This string parameter accepts the complete path of the file being checked out to SourceSafe.
- Comment: This string parameter accepts the comment applied by the user to file check out.

The following Visual Basic code displays a message box with the name of the file that is being checked out, the local path it will be checked out to, and the check out comment applied. If no comment has been added, the checkout is canceled:

```
Function VSSHandler_BeforeCheckOut(ByVal Item As IVSSItem, _
ByVal LocalSpec As String, ByVal Comment As String) As Boolean
   MsgBox ("The file " + LocalSpec + " is about to be checked out to " + _
   Item.Parent.Spec + " with a comment of " + Comment)
   If Comment = "" Then
      VSSHandler_ BeforeCheckOut = False
      Else
      VSSHandler_ BeforeCheckOut = True
   EndIf
End Function
```

If you set the function VSSHandler_BeforeCheckOut to false, the checking out of the file object will not occur.

When you check out a project object, each file to be checked out from the project will fire this event.

BeforeEvent(iEvent As Long, pIItem As VSSItem, Str As String, var)

Currently this event serves no purpose. It may be used in future versions of SourceSafe.

BeforeRename(pIItem As VSSItem, NewName As String)

The BeforeRename event is fired immediately prior to the renaming of a file or project in the SourceSafe database. This event accepts two parameters that are passed to your add-in from Visual SourceSafe:

- pIItem: This VSSItem parameter accepts a VSSItem object representing the file or project that will be renamed in the SourceSafe database.
- NewName: This string parameter accepts the new name to be applied to the file or project item.

The following Visual Basic code displays a message box that contains the current and proposed name of the file or project object. If the proposed name exceeds 10 characters in length, the rename is canceled:

```
Function VSSHandler_BeforeRename(ByVal Item As IVSSItem, _
ByVal NewName As String) As Boolean
   Dim strItemType As String
   If Item.Type = VSSITEM_FILE Then
      strItemType = "file"
   Else
      strItemType = "project"
   End If
   MsgBox ("The " + strItemType + " " + Item.Name +  _
   " is about to be renamed to " + NewName)
   If Len(NewName) > 10 Then
      VSSHandler_ BeforeRename = False
      Else
      VSSHandler_ BeforeRename = True
   EndIf
   End Function
```

If you set the function VSSHandler_BeforeRename to false, the rename of the file or project object will not occur.

BeforeUndoCheckOut(pIItem As VSSItem, Local As String)

The BeforeUndoCheckOut event is fired immediately prior to unchecking out a file item from the SourceSafe database. This event accepts two parameters that are passed to your add-in from Visual SourceSafe:

- **pIItem:** This *VSSItem* parameter accepts a **VSSItem** object representing the file being unchecked out from SourceSafe.
- **Local:** This string parameter accepts the complete path of the file being unchecked from SourceSafe.

The following Visual Basic code displays a message box describing the SourceSafe file path that is being unchecked out and the local folder from which it will be unchecked out. If the user is attempting to uncheck out the file from the root of their C drive, the undo checkout is canceled:

```
Function VSSHandler_BeforeUndoCheckOut(ByVal Item As IVSSItem, _
ByVal LocalSpec As String) As Boolean
   MsgBox ("The file " + Item.Spec + " is about to be " + _
       "unchecked out from " + LocalSpec)
   If Left(LocalSpec, 2) = "C:" Then
      VSSHandler_ BeforeUndoCheckOut = False
      Else
      VSSHandler_ BeforeUndoCheckOut = True
   EndIf
End Function
```

When you undo checkout of a project object, each file that is unchecked out will fire this event.

If you set the function VSSHandler_BeforeUndoCheckOut to false, the undo checkout of the file will not occur.

BeginCommand(unused As Long)

The BeginCommand event is fired when the user initiates any SourceSafe command. The following Visual Basic code displays a message box whenever a command is initiated:

```
Function VSSHandler_BeginCommand(ByVal unused As Long) As Boolean
   MsgBox ("BeginCommand")
   VSSHandler_BeginCommand = True
End Function
```

If you set the function VSSHandler_BeginCommand to false, the command will be canceled.

Unfortunately there is currently no way to distinguish one command from another. Therefore, this event can only be used as an "all or nothing" way of allowing users to access the database. For example, if you wanted to prevent a specific user (or all users for that matter) from modifying the database, you could set VSSHandler_BeginCommand to false.

EndCommand(unused As Long)

The EndCommand event is fired after any SourceSafe command has been completed. The following Visual Basic code displays a message box whenever a command is done:

```
Sub VSSHandler_EndCommand(ByVal unused As Long)
   MsgBox ("EndCommand")
End Sub
```

Properties

VSSDatabase As VSSDatabase (Read-only)

The VSSDatabase property represents an instance of a Visual SourceSafe database. This object exposes the following properties and methods:

Properties:

- **CurrentProject** as String

- **DatabaseName** as String (Read-only)
- **DefaultProjectRights** as Long
- **ProjectRightsEnabled** as Boolean
- **SrcSafeIni** as String (Read-only)
- **User** (Name As String) as VSSUser (Read-only)
- **UserName** as String (Read-only)
- **Users** as IVSSUsers (Read-only)
- **VSSItem** (Spec as String, [Deleted as Boolean = False]) as VSSItem (Read-only)
- **Methods:**
- **AddUser** (User as String, Password as String, Read-only as Boolean) as VSSUser
- **Open** ([SrcSafeIni as String], [Username as String], [Password as String])

The following Visual Basic code displays login information obtained from the database object:

```
Implements IVSSEventHandler
' The next statement creates an instance of a VSSApp Object
Dim WithEvents VSSHandler As VSSApp
Sub IVSSEventHandler_Init(ByVal pIVSS As VSSApp)
   Set VSSHandler = pIVSS
   MsgBox ("User " + pIVSS.VSSDatabase.Username + _
   " has just logged into SourceSafe.")
End Sub
```

For a complete description of these properties and methods, see the "VSSDatabase Object" section later in this article.

Putting It All Together—Trapping Visual SourceSafe Events

Setting Up the Visual Basic Development Environment

If you want to create an application that uses the SourceSafe Automation interface, you need to set a reference to the SourceSafe 6.0 Type Library. If this option doesn't appear in your Visual Basic Reference list, use the Browse command from the References dialog box to locate the file ssapi.dll. By default, this file will reside in the WIN32 folder of your Visual SourceSafe 6.0 installation.

Writing the Code

This section takes you through a step-by-step process of writing a Visual Basic ActiveX DLL that traps events in a SourceSafe database. You may copy this code sample and paste it into a Visual Basic 5.0 ActiveX DLL project to view the sample in action.

Begin by opening a new ActiveX DLL project and setting a reference to the SourceSafe 6.0 Type Library. To trap SourceSafe events, your ActiveX DLL must support the IVSSEventHandler interface. In Visual Basic, this is best accomplished using the IMPLEMENTS statement. You can implement the IVSSEventHandler interface by adding the following code to your Class module:

```
Implements IVSSEventHandler
```

Next we add code that creates an instance of the VSSApp object:

```
Dim WithEvents VSSHandler As VSSApp
```

When SourceSafe is opened, it will create an instance of your add-in and call its Init method. This method is exposed through the IVSSEventHandler interface. Add the following procedure to your Class module to create the Init method:

```
Sub IVSSEventHandler_Init(ByVal pIVSS As VSSApp)
   Set VSSHandler = pIVSS
   ' This optional line displays login information when SourceSafe is opened
   MsgBox ("User " + pIVSS.VSSDatabase.Username + _
   " has just logged into the database located at " + _
   pIVSS.VSSDatabase.SrcSafeIni)
End Sub
```

When Visual Basic implements the IVSSEventHandler interface, it provides its own version of every procedure exposed by the interface. Consequently, our ActiveX DLL must include code for each of these procedures. The following Visual Basic code contains all of the procedures exposed by the interface. It includes the sample procedures used in the "Events" section earlier in this article. Copy this code, and paste it into your Class module:

```
Function VSSHandler_BeforeAdd(ByVal Prj As IVSSItem, ByVal LocalSpec _
As String, ByVal Comment As String) As Boolean
   MsgBox ("The file " + LocalSpec + " is about to be added to " + Prj.Spec _
   + " with a comment of " + Comment)
   VSSHandler_BeforeAdd = True
End Function
Sub VSSHandler_AfterAdd(ByVal Item As IVSSItem, ByVal LocalSpec As _
String, ByVal Comment As String)
   MsgBox ("The file " + LocalSpec + " was just added to " + Item.Parent.Spec
   + " with a comment of " + Comment)
End Sub
Function VSSHandler_BeforeBranch(ByVal Item As IVSSItem, ByVal _
Comment As String) As Boolean
   MsgBox ("The file " + Item.Name + " in the project " + _
   Item.Spec + " is about to be branched with a comment of " + _
   Comment)
   VSSHandler_BeforeBranch = True
End Function
Sub VSSHandler_AfterBranch(ByVal Item As IVSSItem, ByVal Comment _
As String)
   MsgBox ("The file " + Item.Name + " in the project " + _
   Item.Parent.Name + " was just branched with a comment of " + _
   Comment)
End Sub
Function VSSHandler_BeforeCheckIn(ByVal Item As IVSSItem, ByVal _
LocalSpec As String, ByVal Comment As String) As Boolean
   MsgBox ("The file " + LocalSpec + " is about to be checked in to " + _
   Item.Parent.Spec + " with a comment of " + Comment)
   VSSHandler_BeforeCheckIn = True
```

```
End Function
Sub VSSHandler_AfterCheckIn(ByVal Item As IVSSItem, ByVal LocalSpec _
As String, ByVal Comment As String)
   MsgBox ("The file " + LocalSpec + " was just checked in to " + _
   Item.Parent.Spec + " with a comment of " + Comment)
End Sub
Function VSSHandler_BeforeCheckOut(ByVal Item As IVSSItem, ByVal _
LocalSpec As String, ByVal Comment As String) As Boolean
   MsgBox ("The file " + LocalSpec + " is about to be checked out to " + _
   Item.Parent.Spec + " with a comment of " + Comment)
   VSSHandler_BeforeCheckOut = True
End Function
Sub VSSHandler_AfterCheckOut(ByVal Item As IVSSItem, ByVal LocalSpec _
As String, ByVal Comment As String)
   MsgBox ("The file " + LocalSpec + " was just checked out to " + _
   Item.Parent.Spec + " with a comment of " + Comment)
End Sub
Function VSSHandler_BeforeRename(ByVal Item As IVSSItem, ByVal _
NewName As String) As Boolean
   Dim strItemType As String
   If Item.Type = VSSITEM_FILE Then
      strItemType = "file"
   Else
      strItemType = "project"
    End If
    MsgBox ("The " + strItemType + " " + Item.Name + _
    " is about to be renamed to " + NewName)
    VSSHandler_BeforeRename = True
End Function
Sub VSSHandler_AfterRename(ByVal Item As IVSSItem, ByVal OldName _
As String)
   Dim strItemType As String
   If Item.Type = VSSITEM_FILE Then
      strItemType = "file"
   Else
      strItemType = "project"
   End If
   MsgBox ("The " + strItemType + " " + OldName + _
   " was just renamed to " + Item.Name)
End Sub
Function VSSHandler_BeforeUndoCheckOut(ByVal Item As IVSSItem, _
ByVal LocalSpec As String) As Boolean
   MsgBox ("The file " + Item.Spec + " is about to be " + _
   "unchecked out from " + LocalSpec)
   VSSHandler_BeforeUndoCheckOut = True
End Function
Sub VSSHandler_AfterUndoCheckOut(ByVal Item As IVSSItem, ByVal _
LocalSpec As String)
   MsgBox ("The file " + Item.Spec + " was just unchecked out from " + _
   "the folder " + LocalSpec)
End Sub
Function VSSHandler_BeginCommand(ByVal unused As Long) As Boolean
   MsgBox ("BeginCommand")
   VSSHandler_BeginCommand = True
End Function
Sub VSSHandler_EndCommand(ByVal unused As Long)
   MsgBox ("EndCommand")
End Sub
Function VSSHandler_BeforeEvent(ByVal iEvent As Long, ByVal Item _
As IVSSItem, ByVal Str As String, ByVal Var As Variant) As Boolean
```

```
   MsgBox ("BeforeEvent")
   VSSHandler_BeforeEvent = True
End Function

Sub VSSHandler_AfterEvent(ByVal iEvent As Long, ByVal Item As _
IVSSItem, ByVal Str As String, ByVal Var As Variant)
   MsgBox ("AfterEvent")
End Sub
```

Now that we have coded each procedure exposed by the interface, we can compile our DLL. Once this is accomplished, use RegEdit to located the ProgID of the DLL. This can be done by searching the registry for the string *<ProjectName>.<ClassName>* and then opening the ProgID key for the entry. If you used the name Project1 for your DLL and the name Class1 for your Class module, by default the ProgID will be Project1.Class1.

Now we need to let SourceSafe know about our add-in. We accomplish this by modifying or creating the text file ssaddin.ini. This file must reside in the same folder as ssapi.dll. By default, this is the ...\VSS\WIN32 folder. After opening or creating the file, add a reference to our add-in using the syntax:

```
<ProgID>=1
```

For example, if our ProgID is Project1.Class1, we would need to add the line:

```
Project1.Class1=1
```

After adding the reference and closing the file, we are ready to test our add-in. Launch SourceSafe, log on to a database, and add, check out, check in, rename, and undo checkout of some files. You should now be seeing message boxes describing each of these events.

To turn off the add-in, simply remove its reference in the file ssaddin.ini, or set the reference equal to 0 instead of 1.

Note To trap events in SourceSafe, each SourceSafe installation must have its own copy of ssaddin.ini and the ActiveX DLL (or add-in). For example, suppose you have SourceSafe installed on Server A and a client installation of SourceSafe on workstations B and C. The file ssaddin.ini must appear in the ...\VSS\WIN32 folder of A, B, and C, and you must install and register the add-in on each system (A, B, and C).

If you were to add a third SourceSafe installation to the preceding example (workstation D) and not add both of these files, SourceSafe events that occur on workstation D will not be trapped (although events on A, B, and C will continue to be trapped).

Driving a SourceSafe Database

VSSCheckOut Object

The VSSCheckOut object represents a CheckOut of a file from a Visual SourceSafe database. Each Visual SourceSafe file maintains a record of all of its CheckOuts. This record is referred to as its CheckOuts collection. You may access this collection through Automation. This section describes the properties and methods of the VSSCheckOut object.

Properties

Comment As String (Read-only)

The Comment property of the VSSCheckOut object is used to return the comment applied at the time the CheckOut was made.

The following Visual Basic code demonstrates how to iterate through the CheckOuts collection of a file and print each comment (if any) that was applied to the CheckOut:

```
Dim objVSSCheckOut as VSSCheckOut
Dim objVSSFile as VSSItem
<...>
For Each objVSSCheckOut In objVSSFile.CheckOuts
   Debug.Print objVSSCheckOut.Comment
Next
```

The Comment property of a CheckOut object may be set when the file or project's CheckOut method is called.

Date As Date (Read-only)

The Date property of the VSSCheckOut object is used to return the date that the CheckOut occurred.

The following Visual Basic code demonstrates how to iterate through the CheckOuts collection of a file and print the date of each CheckOut of the file:

```
Dim objVSSCheckOut as VSSCheckOut
Dim objVSSFile as VSSItem
<...>
For Each objVSSCheckOut In objVSSFile.CheckOuts
   Debug.Print Str(objVSSCheckOut.Date)
Next
```

The format of the Date property is determined by the regional settings of the operating system and includes both the date and the time. The Date property of a CheckOut object is set when the file or project's CheckOut method is called.

LocalSpec As String (Read-only)

The LocalSpec property of the VSSCheckOut object is used to return the path to which the CheckOut occurred. This may or may not be the Working Folder of the file's parent project. The CheckOut's LocalSpec is set when the file or project's CheckOut method is called.

The following Visual Basic code demonstrates how to iterate through the CheckOuts collection of a file and print the LocalSpec for each CheckOut of the file:

```
Dim objVSSCheckOut as VSSCheckOut
Dim objVSSFile as VSSItem
<...>
For Each objVSSCheckOut In objVSSFile.CheckOuts
   Debug.Print objVSSCheckOut.LocalSpec
Next
```

The LocalSpec property of a CheckOut object returns the complete path of the checked-out folder but does not include the file name. For example, if the file $/MyFile.txt was checked out to C:\WorkingFolder, the LocalSpec property of the CheckOut will return C:\WorkingFolder, not C:\WorkingFolder\MyFile.txt. The LocalSpec property of a CheckOut object is set when the file or project's CheckOut method is called.

Machine As String (Read-only)

The Machine property of the VSSCheckOut object is used to return the machine name from which the CheckOut occurred. Occasionally, a SourceSafe user may check out a file from several different machines. For example, they may check out a file on their main system and then check the same file out on their laptop. This scenario becomes more complex when the user has checked the file out to identical working folders (the user may have checked the file out to C:\Working on both machines). The Machine property of the VSSCheckOut allows you to distinguish these CheckOuts by tagging the VSSCheckOut with the unique machine name.

The following Visual Basic code demonstrates how to iterate through the CheckOuts collection of a file and print the Machine property and CheckOut folder of each CheckOut of the file:

```
Dim objVSSCheckOut as VSSCheckOut
Dim objVSSFile as VSSItem
<...>
For Each objVSSCheckOut In objVSSFile.CheckOuts
   Debug.Print objVSSCheckOut.Machine
   Debug.Print objVSSCheckOut.LocalSpec
Next
```

The Machine property of a CheckOut object is set when the file or project's CheckOut method is called.

Project As String (Read-only)

The Project property of the VSSCheckOut object is used to return the project path from which the CheckOut occurred. In the case of shared files, a SourceSafe user may check out a file from one project and then check it in from another. Automation allows you to identify this situation through use of the Project property.

The following Visual Basic code demonstrates how to iterate through the CheckOuts collection of a file and print the Project property for each CheckOut of the file:

```
Dim objVSSCheckOut as VSSCheckOut
Dim objVSSFile as VSSItem
<...>
For Each objVSSCheckOut In objVSSFile.CheckOuts
   Debug.Print objVSSCheckOut.Project
Next
```

The Project property of a CheckOut object is set when the file or project's CheckOut method is called.

UserName As String (Read-only)

The UserName property of the VSSCheckOut object is used to return the name of the SourceSafe user that created the CheckOut.

The following Visual Basic code demonstrates how to iterate through the CheckOuts collection of a file and print the UserName property of each CheckOut of the file:

```
Dim objVSSCheckOut as VSSCheckOut
Dim objVSSFile as VSSItem
<...>
For Each objVSSCheckOut In objVSSFile.CheckOuts
   Debug.Print objVSSCheckOut.UserName
Next
```

The UserName property of a CheckOut object is set when the file or project's CheckOut method is called.

VersionNumber As Long (Read-only)

The VersionNumber property of the VSSCheckOut object is used to return the version number of the file when the CheckOut occurred. SourceSafe tracks the version number of every file and project within its database. The VersionNumber property is incremented each time the file is updated within the database.

The following Visual Basic code demonstrates how to iterate through the CheckOuts collection of a file and print the VersionNumber property of each CheckOut of the file:

```
Dim objVSSCheckOut as VSSCheckOut
Dim objVSSFile as VSSItem
<...>
For Each objVSSCheckOut In objVSSFile.CheckOuts
   Debug.Print Str(objVSSCheckOut.VersionNumber)
Next
```

The VersionNumber property of a CheckOut object is set when the file or project's CheckOut method is called.

VSSDatabase Object

The VSSDatabase object represents a Visual SourceSafe database. This section describes the properties and methods of the VSSDatabase object.

Properties

CurrentProject As String

The CurrentProject property of the database object is used to set or return the currently selected project in the Visual SourceSafe database as reflected in the current user's ss.ini file. *This property does not change any pointers or references to the VSSItem.* Use the VSSItem property of the database object to set the current file or project.

Each SourceSafe user has their own unique ss.ini file that contains their individual preferences for the SourceSafe Explorer. This file includes a reference to the most recently selected project in the database. The SourceSafe Explorer uses this setting to go to the last project visited by a user when they reopen SourceSafe.

The CurrentProject property may be used to set or return this value through Automation. For example, if a user opens Visual SourceSafe for the first time and visits a project called $/MyProject, the Visual Basic call:

```
objVSSDatabase.CurrentProject = "$/MyProject"
```

would set the CurrentProject setting of the user's ss.ini file to "$/MyProject" once the database object is released. You may retrieve this setting in code and set the current project object as appropriate.

DatabaseName As String (Read-only)

Using the SourceSafe Explorer, the SourceSafe Administrator can assign a default name to a database. The DatabaseName property returns this default database name (if any) assigned by the SourceSafe Administrator. In Visual Basic, this value may be retrieved using the following call:

```
strVar = objVSSDatabase.DatabaseName
```

This property does not assign or return the name of the database assigned by an individual user. If the administrator has not assigned a default database name, this property returns an empty string.

DefaultProjectRights As Long

The SourceSafe Administrator has the ability to add new users to the SourceSafe database Users collection. In Automation, this is accomplished through the AddUser method of the database object. The DefaultProjectRights property is used to set or return the default rights for a new user added to the SourceSafe database.

Note The ProjectRightsEnabled property of the database must be set to true for this property to be effective. Only user Admin can change this property.

The only valid values for DefaultProjectRights are as follows.

Value	Rights
0	No rights
1	Read rights
3	Read /check out/check in rights
7	Read /check out/check in/add/rename rights
15	Read /check out/check in/add/rename/destroy rights

Each of these values may be created through a combination of predefined constants exposed in the Automation interface. These constants are as follows.

Constant	Value
VSSRIGHTS_READ	1
VSSRIGHTS_CHKUPD	2
VSSRIGHTS_ADDRENREM	4
VSSRIGHTS_DESTROY	8
VSSRIGHTS_ALL	15

You can programmatically create invalid values for DefaultProjectRights (values other than those just listed). An example is:

```
objVSSDatabase.DefaultProjectRights = _
VSSRIGHTS_DESTROY + VSSRIGHTS_READ
```

This combination (a value of 9) attempts to tell SourceSafe to give a new user destroy rights without the user having add/rename/check out/check in rights. These invalid combinations are not supported.

Note Invalid DefaultProjectRights combinations do not generate run-time errors but will produce unexpected results.

ProjectRightsEnabled As Boolean

The ProjectRightsEnabled property of the database object is used to set or return if project rights are enabled for all SourceSafe users. This Boolean value may be set with the following Visual Basic command:

```
objVSSDatabase.ProjectRightsEnabled = True
```

or retrieved with this command:

```
boolVar = objVSSDatabase.ProjectRightsEnabled
```

Only user Admin can access this property. In Visual Basic, if any other user attempts to access this property, the run-time error -2147166522 is generated. If you change this setting through the Automation interface and close the database object, the SourceSafe Explorer and the Admin utility will use the new setting the next time they are launched.

SrcSafeIni As String (Read-only)

The SrcSafeIni property of the database object is used to return the complete path to the current SourceSafe database's srcsafe.ini file. If an instance of the database object has been created in Visual Basic, you may use the call:

```
strVar = objVSSDatabase. SrcSafeIni
```

to return the current srcsafe.ini path at any time. To change the current SrcSafeIni property, you must use the Open method of the VSSDatabase object and create an instance of a new database object.

The Automation interface allows you to open multiple databases simultaneously; however, a specific database object may reference only one database at one time. The following Visual Basic code demonstrates how to close a database object:

```
Set objVSSDatabase = Nothing
```

The SrcSafeIni property includes the file name srcsafe.ini. Using the Open method of the database object with a SrcSafeIni variable of "C:\VSS" will fail while the value "C:\VSS\SRCSAFE.INI" will succeed.

See the "Open Method" topic of the "VSSDatabase Object" section for more information.

User (Name As String) As VSSUser (Read-only)

Every SourceSafe database has one or more users (such as Admin and Guest). In Automation, each user may be created as an object type known as a VSSUser. The VSSUser object has several properties and methods, such as Name, Password, Delete, and RemoveProjectRights. (For a complete description of these and other properties and methods see the "VSSUser Object" section in this article.) The User property of the database object may be used to reference any VSSUser in the database object.

The User property accepts a single parameter:

Name. This required string variable contains the name of the user you want to reference.

For example, the following Visual Basic code demonstrates how to create an instance of a

VSSUser object and print the user's name:

```
Dim objVSSUser As VSSUser
Dim strUserName As String
strUserName = "Guest"
<...>
Set objVSSUser = objVSSDatabase.User(strUserName)
Debug.Print objVSSUser.Name
```

Here, another example demonstrates how to set the password for a SourceSafe user:

```
Dim objVSSUser As VSSUser
Dim strUserName As String
strUserName = "Guest"
<...>
Set objVSSUser = objVSSDatabase.User(strUserName)
objVSSUser.Password = strVar
```

To change the password for any given user you must be logged on as user Admin or logged on as the specific user for whom you wish to change the password.

You may add or remove users from the Users collection using the objVSSDatabase.AddUser or the VSSUser.Delete methods, respectively.

UserName As String (Read-only)

The UserName property of the database object is used to return the name of the Visual SourceSafe user currently logged into the SourceSafe database. In Visual Basic, you may use the call:

```
strVar = objVSSDatabase. UserName
```

to return the current user's logon name at any time. To change the currently logged-on user, you must use the Open method of the VSSDatabase object and create an instance of a new database object.

The Automation interface allows you to open multiple databases simultaneously; however, a specific database object may reference only one database at one time. The following Visual Basic code demonstrates how to close a database object:

```
Set objVSSDatabase = Nothing
```

See the "Open Method" topic of the "VSSDatabase Object" section for more information.

Users As IVSSUsers (Read-only)

The Users property of the database object is used to reference all users within the SourceSafe database (also known as the Users Collection object). The following Visual Basic code demonstrates how to list all users in the current Visual SourceSafe database:

```
Dim objVSSUser As VSSUser
<...>
For Each objVSSUser In objVSSDatabase.Users
   Debug.Print objVSSUser.Name
Next User
```

You may add or remove users from the Users collection using the objVSSDatabase.AddUser or the VSSUser.Delete methods, respectively.

VSSItem (Spec As String, [Deleted As Boolean = False]) As VSSItem (Read-only)

The VSSItem property of the database object is used to return information on a created instance of a VSSItem object. A VSSItem object represents either a SourceSafe project or a SourceSafe file (either of which may be deleted or non-deleted). You can use the Type property of the VSSItem to determine what type of object (project or file) the VSSItem represents.

The VSSItem object has many properties and methods, such as Deleted and Binary or CheckIn and CheckOut. For a complete listing of these properties and methods, please see the "VSSItem Object" section in this article.

The database object's VSSItem property accepts two parameters:

- **Spec:** This required string variable is used to retrieve the *complete* SourceSafe path of the **VSSItem**. Examples are "$/" and $/MyProject/MyFile.txt."
- **Deleted:** This optional Boolean variable is used to tell Automation whether to look for deleted or non-deleted items in the current call. If this parameter is set to false, only non-deleted items are included. When it is set to true, only deleted items are included. The default for this parameter is false.

The following Visual Basic sample iterates each item in a SourceSafe project and prints the object's name, ignoring both deleted items and subprojects:

```
Dim objVSSProject As VSSItem
Dim objVSSObject As VSSItem
<...>
For Each objVSSObject In objVSSProject.Items(False)
   If objVSSObject.Type = VSSITEM_FILE Then
      Debug.Print ("I am a non-deleted file named: " + _
      objVSSObject.Name)
   End If
Next
```

Conversely, to list all deleted subprojects in a given project the following code may be used:

```
Dim objVSSProject As VSSItem
Dim objVSSObject As VSSItem
<...>
For Each objVSSObject In objVSSProject.Items(True)
   If objVSSObject.Type = VSSITEM_PROJECT Then
      Debug.Print ("I am a deleted subproject named: " + _
      objVSSObject.Name)
   End If
Next
```

The following Visual Basic code demonstrates how to create an instance of a new VSSItem object:

```
Dim objVSSFile As VSSItem
Dim strItemPath As String
<...>
Set objVSSFile = objVSSDatabase.VSSItem(strItemPath)
```

Note that the string variable ItemPath may be used to represent either a SourceSafe project or a file item. For example, both

```
strItemPath = "$/"
strItemPath = "$/MyProject/MyFile.txt"
```

are equally valid in the preceding example.

Methods

AddUser (User As String, Password As String, ReadOnly As Boolean) As VSSUser

The SourceSafe Administrator has the ability to add new users to the current SourceSafe database. In Automation, this is accomplished through the AddUser method of the database object. This method adds a new user to the current database object's Users collection. This method has the following parameters:

- **User:** This required string parameter contains the unique name of a new user to add to the database. If you attempt to add a user that already exists, the Visual Basic run-time error -2147166525 will be generated.
- **Password:** This required string is used to specify the password for the new user. If you attempt to give the user an invalid password the Visual Basic run-time error -2147352566 will be generated. An empty string will set the password to nothing.
- **ReadOnly:** This required Boolean variable is used to specify if the new user has read-only rights to the database. If this property is set to true, it overrides the **DefaultProjectRights** settings of the database. For example, if **DefaultProjectRights** is set to VSSRIGHTS_ALL, and a new user is added to the database with the *ReadOnly* parameter set to true, the new user will have read-only rights to the database, not VSSRIGHTS_ALL. If the *ReadOnly* parameter is set to false, the new user is given whatever rights are assigned to the value **DefaultProjectRights.**

The following Visual Basic code demonstrates how to add a new user to a SourceSafe database:

```
Dim objUserToAdd As VSSUser
Dim strName as String
Dim strPassword As String
Dim Rights As Boolean
<...>
Set objUserToAdd = objVSSDatabase.AddUser(strName, strPassword,  _
Rights)
```

The AddUser method may be called only by user Admin. In Visual Basic, if any other user attempts to call the AddUser method, the run-time error -2147166522 will be generated.

Open ([SrcSafeIni As String], [Username As String], [Password As String])

The Open method of the database object is used to create an instance of a database object. The Automation interface allows you to open multiple databases simultaneously, however a specific database object may reference only one database at one time. The Open method accepts the following three parameters:

- **SrcSafeIni:** This optional string parameter is used to specify the complete path to the current SourceSafe database's srcsafe.ini file. The *SrcSafeIni* parameter includes the file name srcsafe.ini. Using the **Open** method of the database object with a SrcSafeIni variable of "C:\VSS" will fail, while the value "C:\VSS\SRCSAFE.INI" will succeed. In Visual Basic, if an invalid path is passed, the run-time error -2147167977 will be generated. When this parameter is not passed, SourceSafe looks for the srcsafe.ini in the following order:

- Search for srcsafe.ini in the directory where ssapi.dll is located.
- Search for srcsafe.ini in each directory of the path to ssapi.dll. In other words, if ssapi.dll is located in C:\Folder1\Folder2\Folder3\SSAPI.DLL, Folder3, Folder2, Folder1 and C:\ are searched (in that order).
- Search for srcsafe.ini in the location indicated by the named value "API Current Database" in the registry key HKEY_LOCAL_MACHINE\Software\Microsoft\SourceSafe.
- Search for srcsafe.ini in the location indicated by the named value "SCCProviderPath" in the registry key HKEY_LOCAL_MACHINE\Software\Microsoft\SourceSafe.

- **UserName:** This optional string parameter is used to specify the name of the Visual SourceSafe user currently logging into the SourceSafe database. In Visual Basic, if an invalid UserName is passed, the run-time error -2147166526 will be generated. The default value of this parameter is the name of the user logged into the Windows session.
- **Password:** This optional string parameter is used to specify the password of the Visual SourceSafe user attempting to log into the SourceSafe database. If the user has no password, you must pass an empty string. The default value of this parameter is an empty string. In Visual Basic, if an invalid password is passed, the run-time error -2147166519 will be generated.

The following Visual Basic code demonstrates using the Open method to create an instance of a database object:

```
Dim strSrcSafeIni As String
Dim strUserName As String
Dim strPassword As String
strSrcSafeIni = "C:\VSS\SRCSAFE.INI"
strUserName = "Guest"
strPassword = ""
<...>
' Attempt to log into SourceSafe
objVSSDatabase.Open strSrcSafeIni, strUserName, strPassword
```

The following Visual Basic code demonstrates how to close a created instance of a database object:

```
Set objVSSDatabase = Nothing
```

Note Any user may call the Open method.

VSSFileStatus

The predefined VSSFileStatus constants may be used to determine the CheckOut status of a file object. These are used with the IsCheckedOut property of the file object and are defined as follows.

Constant	Value
VSSFILE_NOTCHECKEDOUT	0
VSSFILE_CHECKEDOUT	1
VSSFILE_CHECKEDOUT_ME	2

The following Visual Basic code demonstrates how to determine the CheckOut status of a file. The sample uses the Select Case function against the IsCheckedOut property of the file object to determine whether it is checked out, checked out by multiple users, or not checked out:

```
Dim objVSSFile As VSSItem
Dim objVSSCheckOut As VSSCheckOut
Dim strUserName As String
Dim strOtherUser As String
Dim CheckOutMulti As Boolean
<...>
' Set UserName to the current VSS User
strUserName = objVSSDatabase.Username
Select Case objVSSFile.IsCheckedOut

   ' File is checked out by current user
   Case VSSFILE_CHECKEDOUT_ME

      ' Check to see if the file is checked out by
      ' current user only, or if the file is
      ' checked out by multiple users
      For Each objVSSCheckOut In objVSSFile.CheckOuts
         If objVSSCheckOut.UserName <> strUserName Then
            MsgBox ("File is checked out by current user and others.")
            CheckOutMulti = True
            Exit For
         EndIf
      Next
      If Not CheckOutMulti Then
         MsgBox ("File is checked out by current user only.")
      EndIf
   ' File is checked out by another user
   Case VSSFILE_CHECKEDOUT

      ' Find the username. Since this will be the first item in the CheckOuts
      ' Collection, get the name on the first pass and then exit for loop
      For Each objVSSCheckout In objVSSFile.Checkouts
         strOtherUser = objVSSCheckout.UserName
         Exit For
      Next
      ' Check for multiple CheckOuts
      For Each objVSSCheckOutOther In objVSSFile.CheckOuts
         If objVSSCheckoutOther.UserName <> strOtherUser Then
            MsgBox("File is checked out by more than one user and none _
            of  them is the current user.")
            CheckOutMulti = True
            Exit For
         End If
      Next
      If Not CheckOutMulti Then
```

```
            MsgBox ("File is checked out by one other user only.")
        EndIf

    ' File is not checked out
    Case Else
        MsgBox ("File is not checked out.")

End Select
```

Note The VSSFileStatus constants are not cumulative. In other words, if a file is checked out by the current user *and* checked out by another user, the IsCheckedOut property of the file object returns:

```
VSSFILE_CHECKEDOUT_ME
```

not

```
VSSFILE_CHECKEDOUT + VSSFILE_CHECKEDOUT_ME
```

VSSFlags (Flags Used within Automation)

The following predefined VSSFlags may be used to set or return a variety of Visual SourceSafe settings. Flags marked as default are used when no flag is specified. Flags that are used for a common setting are grouped together.

By default, SourceSafe automatically detects whether an added file is text or binary. It accomplishes this by checking the file for embedded null characters. Use these flags with the Add method of a project object to customize this behavior.

- **Const VSSFLAG_BINBINARY**
- When this flag is set, SourceSafe sets the added file type to binary.
- **Const VSSFLAG_BINTEST** (default)
- When this flag is set, SourceSafe auto-detects the added file's file type.
- **Const VSSFLAG_BINTEXT**
- When this flag is set, SourceSafe sets the added file type to text.

SourceSafe allows text files to be checked out by more than one user. Use these flags with the CheckOut method of a project or file object to customize this behavior. The Allow Multiple CheckOuts option (in the SourceSafe Admin utility) must be set for this flag to be effective. Currently, the Allow Multiple CheckOuts option is not exposed in Automation.

- **Const VSSFLAG_CHKEXCLUSIVENO** (default if **Allow Multiple CheckOuts** is enabled)
- When this flag is set, SourceSafe allows the item to be checked out by multiple users. If the **Allow Multiple CheckOuts** option in ssadmin.exe is not set, this flag is ignored, and all files are checked out exclusively.
- **Const VSSFLAG_ CHKEXCLUSIVEYES** (default if **Allow Multiple CheckOuts** is disabled)

- When this flag is set, SourceSafe prevents the item from being checked out by multiple users. If the **Allow Multiple CheckOuts** option in ssadmin.exe is not set, this flag is ignored, and all files are checked out exclusively.

SourceSafe allows the user to specify how it determines if the local copy of a file is up to date. Use these flags with the CheckIn and UnCheckOut methods of a project or file object to customize this behavior.

- **Const VSSFLAG_CMPCHKSUM** (default)
- This flag is used to tell SourceSafe to compare files through use of a checksum that it stores internally (this is the recommended method).
- **Const VSSFLAG_CMPFAIL**
- This flag is used to tell SourceSafe to always assume the local file is out of date.
- **Const VSSFLAG_CMPFULL**
- This flag is used to tell SourceSafe to compare the full contents of the local file to the SourceSafe copy.
- **Const VSSFLAG_CMPTIME**
- This flag is used to tell SourceSafe to compare files through use of the file's TimeStamp.

When calling the Add, UndoCheckOut, or CheckIn methods of an object, SourceSafe allows the user to customize whether the local file(s) are deleted. Use these flags to customize this behavior.

- Const VSSFLAG_DELNO (default)

  When this flag is set, the local file(s) will not be deleted.
- **Const VSSFLAG_ DELYES**

  When this flag is set, the local file(s) are deleted.
  - **Const VSSFLAG_DELNOREPLACE**

  When the checkout of a file is undone, SourceSafe allows the user to customize how the local file is handled. When this flag is set, the local file is left in its current condition with the read-only flag set to true. This flag should only be used with the UndoCheckOut method.

SourceSafe allows the user to determine whether a specific file retains its historical versions. Use these flags with the Add method to customize this behavior.

- **Const VSSFLAG_DELTANO**

When this flag is set, the file will not retain its historical versions.

- **Const VSSFLAG_DELTAYES** (default)

  When this flag is set, the file will retain its historical versions.

SourceSafe can be told to append an end-of-line character whenever it retrieves a text file that does not already end with one. These flags are used to customize what type of end-of-line character is added. If no flags are used, the end-of-line character is not modified. Use these flags with the CheckOut, Get, and Branch methods.

Note SourceSafe will change the default end-of-line character based on the operating system it is running on. The values described here apply to the Intel family of computers.

- **Const VSSFLAG_EOLCR**

  When this flag is set, SourceSafe will append a carriage return to the end of all text files that do not already end in one.
- **Const VSSFLAG_EOLCRLF**

  When this flag is set, SourceSafe will append a carriage return and line feed to the end of all text files that do not already end in one.
- **Const VSSFLAG_EOLLF**

  When this flag is set, SourceSafe will append a line feed to the end of all text files that do not already end in one.

SourceSafe allows the user to determine whether a specific SourceSafe command acts on a project based on its working folder or its current folder. Use these flags with the Get, CheckOut, CheckIn, UndoCheckOut, and Branch methods to customize this behavior.

- **Const VSSFLAG_FORCEDIRNO**

  When this flag is set, SourceSafe commands act on the current folder.
- **Const VSSFLAG_FORCEDIRYES**(default)

  When this flag is set, SourceSafe commands act on the working folder.

When a file is checked out, SourceSafe allows the user to prevent the local copy from being copied to the CheckOut folder. Use these flags to customize this behavior.

- **Const VSSFLAG_GETNO**

  When this flag is set, the CheckOut occurs, but the local file(s) in the working folder or VSSItem.LocalSpec are not replaced.
- **Const VSSFLAG_GETYES** (default)

  When this flag is set, the CheckOut(s) occur and the local file(s) in the working folder or VSSItem.LocalSpec are replaced.

When creating an instance of a project object's Versions collection, you may choose to exclude its file histories. Use this flag to customize this setting.

- **Const VSSFLAG_HISTIGNOREFILES**

  This flag may be used when creating an instance of the Versions collection of a project object. If this flag is set, file CheckIns are excluded from the current collection. The following Visual Basic code demonstrates use of this flag:

  ```
  Set objVersionsCollection = _
  objVSSObject.Versions(VSSFLAG_HISTIGNOREFILES)
  ```

When a file is checked in, SourceSafe allows the user keep the file(s) checked out. Use these flags to customize this setting.

- **Const VSSFLAG_KEEPNO** (default)

  When this flag is set, the CheckIn occurs. (The local file is checked in and set to read-only.)
- **Const VSSFLAG_KEEPYES**

  When this flag is set, the CheckIn occurs, and the file(s) remain checked out. (The local file(s) are checked in and remain read/write.)

Many methods exposed in the Automation interface may be called recursively. Use these flags with the CheckIn, CheckOut, Get, or Share methods of a project object to customize this behavior. These flags may also be used when creating an instance of the Versions collection of a project.

- **Const VSSFLAG_RECURSNO** (default)

  When this flag is set, the project is acted on non-recursively. This flag has no effect on file objects.
- **Const VSSFLAG_RECURSYES**

  When this flag is set, the project is acted on recursively. This flag has no effect on the file objects.

SourceSafe allows you to specify how the replacements of local files are handled. These flags allow you to customize this behavior. The following flags may be used with the CheckOut, CheckIn, Get, and UndoCheckOut methods of a file or project object. They may also be used with the Branch method of a file object.

- **Const VSSFLAG_REPASK**

  Currently this flag serves no purpose. It may be used in future versions of SourceSafe.

- **Const VSSFLAG_REPMERGE**

  When this flag is set, SourceSafe will merge files together that have been simultaneously modified by multiple users.

- **Const VSSFLAG_ REPREPLACE**

  When this flag is set, the local file is replaced with the most recent copy from the database.

- **Const VSSFLAG_REPSKIP**

  When this flag is set, SourceSafe will not replace local files that are writable.

SourceSafe allows you to specify what TimeStamp to apply to a local file. Use these flags with the CheckOut, Get, or Branch methods to customize this setting.

- **Const VSSFLAG_TIMEMOD**

  When this flag is set, the TimeStamp of the local file is set to the file's last modification date and time.

- **Const VSSFLAG_TIMENOW** (default)

  When this flag is set, the TimeStamp of the local file is set to the current date and time.

- **Const VSSFLAG_TIMEUPD**

  When this flag is set, the TimeStamp of the local file is set to the date and time that the file was last checked in to the database.

When a user checks in a file or project object and the item(s) have not changed, SourceSafe allows the user to customize how the CheckIn is handled. The choices are:

Check In

-or-

Undo CheckOut

Use these flags with the CheckIn method to customize this setting:

- **Const VSSFLAG_UPDASK**

  Currently this flag serves no purpose. It may be used in future versions of SourceSafe.

- **Const VSSFLAG_UPDUNCH** (default)

  When this flag is set, SourceSafe will undo checkout of unchanged files.

- **Const VSSFLAG_UPDUPDATE**

  When this flag is set, SourceSafe will check in unchanged files.

The following flags serve no purpose but may be used in future versions of SourceSafe.

- **Const VSSFLAG_USERRONO**

  Currently this flag serves no purpose. It may be used in future versions of SourceSafe.
- **Const VSSFLAG_USERROYES**

  Currently this flag serves no purpose. It may be used in future versions of SourceSafe.

VSSItem Object

The VSSItem object represents a file or project item within the SourceSafe database. Before a file or project can be manipulated through Automation, a VSSItem representing that item must be created. The Automation interface allows you to create multiple VSSItems simultaneously, however a specific VSSItem object may reference only one file or project at a time. This section discusses the properties and methods of the VSSItem object.

Properties

Binary As Boolean

Every file in a SourceSafe database is marked as either binary or text. The Binary property of the VSSItem may be used to set or retrieve this setting. If SourceSafe has the file marked as binary, this property will return true, otherwise it will return false. The following Visual Basic code demonstrates how to determine if SourceSafe has marked a file as binary or text:

```
Dim objVSSFile As VSSItem
<…>
If objVSSFile.Binary Then
   MsgBox(objVSSFile.Name + " is marked as binary.")
Else
   MsgBox(objVSSFile.Name + " is marked as text.")
EndIf
```

This next example shows how to mark a file as text:

```
Dim objVSSFile As VSSItem
<…>
objVSSFile.Binary = False
```

The Binary property applies to file objects only. In Visual Basic, if you attempt to access the Binary property of a project object, the run-time error -2147210888 will be generated.

CheckOuts As IVSSCheckOuts (Read-only)

Each Visual SourceSafe file maintains a record of all its CheckOuts. This record is referred to as its CheckOuts collection. The CheckOuts property of the VSSItem represents the CheckOuts collection of the object.

The CheckOuts collection may be used for many purposes, one of which is to find the CheckOut folder of a file that is checked out. The following Visual Basic code demonstrates how this may be accomplished:

```
Dim objVSSCheckOut As VSSCheckOut
Dim objVSSFile As VSSItem
Dim strCheckOutFolder As String
<...>
' File is checked out to the current user
If VSSItem.IsCheckedOut = VSSFILE_CHECKEDOUT_ME Then
' Find where it is checked out to
For Each objVSSCheckout In objVSSFile.Checkouts
          strCheckOutFolder = objVSSCheckOut.LocalSpec
   Exit For
   Next
End If
```

The preceding example assumes the first CheckOut in the CheckOuts collection is the one we are looking for. If the user has the file checked out once, this assumption is valid. However, if the file is checked out by the current user to multiple machines, we would need to add additional code that verifies the property objVSSCheckOut.Machine (or some other unique property). When you iterate through the CheckOuts collection, you move chronologically from the most recent CheckOut to most ancient.

In summary, to manipulate a specific CheckOut in the CheckOuts collection you must iterate through the collection until the desired CheckOut is located. You can verify whether a particular CheckOut is the one you want by validating one or more of its properties (Comment, Date, LocalSpec, Machine, Project, UserName, and VersionNumber).

A new CheckOut item is created each time a file object is checked out. If the file is unchecked out with the flag VSSFLAGS_UPDUNCH, the record is removed.

The CheckOuts property applies to file objects only. In Visual Basic, if you attempt to access the CheckOuts property of a project object, the run-time error -2147210888 is generated.

Deleted As Boolean

SourceSafe allows the user to delete a file or project item. SourceSafe retains deleted items until they are purged or recovered. The Deleted property of a VSSItem is used to set or retrieve the current deleted state of the object. The following Visual Basic code demonstrates how to determine if the VSSItem is deleted:

```
Dim objVSSObject As VSSItem
<...>
If objVSSObject.Deleted Then
   MsgBox(objVSSObject.Name + " is deleted.")
Else
   MsgBox(objVSSObject.Name + " is not deleted.")
EndIf
```

This next example shows how to mark an item as deleted:

```
Dim objVSSObject As VSSItem
<...>
objVSSObject.Deleted = True
```

The Deleted property applies to both file and project items. When a project is deleted, all of its files and subprojects are deleted as well.

IsCheckedOut As Long (Read-only)

The IsCheckedOut property of the VSSItem may be used to determine the CheckOut status of a file object. Automation exposes several predefined constants that may be used in conjunction with this property and are defined as follows.

Constant	Value
VSSFILE_NOTCHECKEDOUT	0
VSSFILE_CHECKEDOUT	1
VSSFILE_CHECKEDOUT_ME	2

The following Visual Basic sample code demonstrates the use of this property:

```
Dim objVSSFile As VSSItem
Dim objVSSCheckOut As VSSCheckOut
Dim strUserName As String
Dim strOtherUser As String
<...>
' Set UserName to the current VSS User
strUserName = objVSSDatabase.Username
Select Case objVSSFile.IsCheckedOut

   ' File is checked out by current user
   Case VSSFILE_CHECKEDOUT_ME
      MsgBox ("File is checked out by " + strUserName  + ".")
   ' File is checked out by another user
   Case VSSFILE_CHECKEDOUT

      ' Find the username. Since this will be the first item in the CheckOuts
      ' Collection, get the name on the first pass and then exit for loop
      For Each objVSSCheckOut In objVSSFile.CheckOuts
         strOtherUser = objVSSCheckOut.UserName
         Exit For
      Next
      MsgBox ("File is checked out by " + strOtherUser + ".")
   ' File is not checked out
   Case Else
      MsgBox ("File is not checked out.")

End Select
```

The preceding sample assumes that multiple CheckOuts are not enabled for the database. For an example that checks for multiple CheckOuts, see the "VSSFileStatus" section of this article.

The IsCheckedOut property applies to file objects only. If you attempt to access the IsCheckedOut property of a project object, the run-time error -2147210888 will be generated.

IsDifferent(Local As String) As Boolean (Read-only)

The IsDifferent property of the VSSItem allows the developer to compare a SourceSafe file against a local file. The IsDifferent property accepts a single parameter:

Local: This required string parameter contains the complete local path of the file to compare the VSSItem against.

If there is no difference between the VSSItem and the local file the IsDifferent property is false, otherwise it is true.

The IsDifferent property can only be used to compare a created instance of a VSSItem against a local file. If you want to compare one VSSItem against another, you must create a local copy of the object using the Get or CheckOut methods first. The following Visual Basic code demonstrates the use of the IsDifferent property by comparing a SourceSafe file object against a local file and displaying the results:

```
Dim objVSSFile As VSSItem
Dim strFileType As String
Dim FilesDiffer As Boolean
Dim Response As Long
Dim strFilePath As String
<...>
' Check to see if file is binary or text
If objVSSFile.Binary Then
   strFileType = "binary"
Else
   strFileType = "text"
End If
' Compare the files
FilesDiffer = objVSSFile.IsDifferent(Local:= strFilePath + objVSSFile.Name)
' Display results
If FilesDiffer Then
   Response = MsgBox("Local " + FileType + " file '" + objVSSFile.Name + _
   "' is different from the SourceSafe version.", vbInformation)
Else
   Response = MsgBox("Local " + FileType + " file '" + objVSSFile.Name + _
   "' is identical to the SourceSafe version.", vbInformation)
End If
```

The IsDifferent property applies to file objects only. If you attempt to access the IsDifferent property of a project object, the run-time error -2147210888 will be generated.

Items([IncludeDeleted As Boolean]) As IVSSItems

The Items property is used to reference the collection of VSSItems within a project object. Each SourceSafe project may contain up to four types of objects. These are:

- Deleted files
- Deleted projects
- Non-deleted files
- Non-deleted projects

The Items property accepts a single Boolean parameter: *IncludeDeleted*. This optional Boolean parameter is used to indicate if deleted items are to be included in the current collection. The default value of this parameter is false.

The following Visual Basic sample code demonstrates how to iterate through all of the non-deleted items in a project and display the name and type of each item:

```
Dim objVSSProject  As VSSItem
Dim objVSSObject As VSSItem
Dim strObjType As String
<...>
' Iterate through each item of the project (false means ignore deleted)
For Each objVSSObject In objVSSProject.Items(False)
' Item is a file
If objVSSObject.Type = VSSITEM_FILE Then
   strObjType = "file"
Else
   strObjType = "project"
EndIf
' Display message
MsgBox("Current item is: " + objVSSObject.Name + " and is a " + _
strObjType + ".")
Next
```

The Items property applies to project objects only. If you attempt to access the Items property of a file object, the run-time error -2147210887 will be generated.

Links As IVSSItems (Read-only)

Visual SourceSafe allows files to be shared between multiple projects. When a file is shared, any changes to the file are propagated to all of the projects that share it. The collection of projects that share a specific file is known as the file's links.

The Links property of a VSSItem is used to reference the collection of VSSItems that form the links for that item. Every VSSItem (including project objects) has at least one link (which is itself). The following Visual Basic code demonstrates creating a list of the links for a specific file:

```
Dim objVSSFile As VSSItem
Dim objVSSLinkItem As VSSItem
Dim strLinkList As String
<...>
' Create a list of the links
For Each objVSSLinkItem In objVSSFile.Links
   strLinkList = strLinkList + objVSSLinkItem.Parent.Spec + vbCrLf
Next
' Display the results
MsgBox("Links for " + objVSSFile.Name + vbCrLf + strLinkList
```

The list created in the preceding sample includes the link to itself. To exclude this one could change the code to:

```
   For Each objVSSLinkItem In objVSSFile.Links
      If objVSSLinkItem.Spec <> objVSSFile.Spec Then strLinkList = _
      strLinkList + objVSSLinkItem.Parent.Spec + vbCrLf
   Next
```

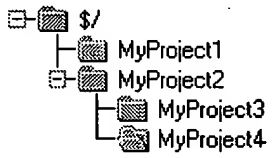
This next example shows how to determine if a file object is shared:

```
Dim objVSSFile As VSSItem
Dim objVSSLinkItem As VSSItem
Dim LinkCount As Integer
Dim Shared As Boolean
<...>
' Check to see if file is shared
For Each objVSSLinkItem In objVSSFile.Links
   LinkCount = LinkCount + 1
   If LinkCount = 2 Then Shared = True
   Exit For
Next
If Shared Then
   MsgBox("File " + objVSSFile.Name + " is shared.")
Else
   MsgBox("File " + objVSSFile.Name + " is not shared.")
EndIf
```

The Links property applies to both file and project objects. While a file object may have one or more items in its Links property, a project object will always have just one (which is itself).

LocalSpec As String

The LocalSpec property is used to set or return the working folder for a VSSItem. This property returns an empty string if there is no working folder set for the item. If the LocalSpec property is set for a project object, all of the project's files and subprojects will recursively inherit this property unless the LocalSpec property of the subproject has already been set. For example, let's say you have the following project structure:



With the following working folders set (the LocalSpec property of $/MyProject2/MyProject4 is set to "C:\Work1," all others are not set).

Project	**Working folder**
$/	None
$/MyProject1	None
$/MyProject2	None
$/MyProject2/MyProject3	None
$/MyProject2/MyProject4	C:\Work1

If you set the LocalSpec property of $/ to "C:\Work2", LocalSpec inheritance would create the following structure.

Project	Working folder
$/	C:\Work2
$/MyProject1	C:\Work2\MyProject1
$/MyProject2	C:\Work2\MyProject2
$/MyProject2/MyProject3	C:\Work2\MyProject2\MyProject3
$/MyProject2/MyProject4	C:\Work1

Note that the subprojects of $/ recursively inherit their LocalSpec property, except where the property has already been set.

The following Visual Basic code checks to see if a project object has a working folder set:

```
Dim objVSSProject  As VSSItem
Dim strWorkingFolder As String
<...>
strWorkingFolder = objVSSProject.LocalSpec
If strWorkingFolder = "" Then
   MsgBox("No working folder is set for this project.")
Else
   MsgBox("The working folder is set for this project.")
EndIf
```

In this next example we will set the working folder:

```
Dim objVSSProject  As VSSItem
Dim strWorkingFolder As String
<...>
objVSSProject.LocalSpec = strWorkingFolder
```

To clear a working folder, set the project's LocalSpec property to an empty string.

The LocalSpec property applies to both file and project objects. However, you may set the LocalSpec property of a project object but not a file object. To change the LocalSpec property of a file object, you must set its parent project's LocalSpec property.

When retrieving the LocalSpec property of a file object, it will return the working folder path and the file name. For example, if $/MyProject has a working folder of C:\MyFolder, the LocalSpec property of $/MyProject would return C:\MyFolder, however the LocalSpec property of $/MyProject/MyFile.txt would return C:\MyFolder\ MyFile.txt.

If you attempt to set the LocalSpec property of a file object, the run-time error -2147210887 will be generated.

Note If you set the LocalSpec property to an invalid or non-existent path, no run-time error is generated.

Name As String

The Name property of the VSSItem is used to set or return the name of the object. This property does not include the SourceSafe path. For example, the Name property of

$/MyProject/MyFile.txt is MyFile.txt. The following Visual Basic code demonstrates how to retrieve and set the Name property:

```
Dim objVSSObject  As VSSItem
Dim strNewName As String
<...>
MsgBox("This item is named " + objVSSObject.Name)
' Rename the item
objVSSObject.Name = strNewName
MsgBox("This item is now named " + objVSSObject.Name)
```

The Name property applies to both file and project objects. In Visual Basic, the run-time error -2147166411 is generated if you attempt to rename the root project. The run-time error -2147166572 is generated if you attempt to rename a file or project to the name of an existing file or project.

Parent As VSSItem (Read-only)

The Parent property of the VSSItem is used to reference the complete SourceSafe path of the parent of a file or project object. The parent of a VSSItem is the project in which it resides. The following Visual Basic code demonstrates using the Parent property:

```
Dim objVSSFile As VSSItem
<...>
MsgBox("The parent of " + objVSSFile.Name + " is named " + _
objVSSFile.Parent.Name
```

The Parent property applies to both file and project objects. An attempt to reference the parent of the root project will return an empty string.

Spec As String (Read-only)

The Spec property of the VSSItem contains the SourceSafe path to the file or project object. The Spec property of a file object includes the file name. For example, the Spec property of $/MyFile.Txt is $/MyFile.Txt, not $/. The following Visual Basic code uses the Spec property to print out the complete SourceSafe path of each item in a project:

```
Dim objVSSItem As VSSItem
Dim objVSSProject As VSSItem
<...>
' Iterate through each item in the project object
For Each objVSSItem In objVSSProject.Items
   Debug.Print objVSSItem.Spec
Next
```

The Spec property applies to both file and project objects.

Type As Long (Read-only)

The Type property of the VSSItem may be used to determine whether a specific item is a file or project. Automation exposes two predefined constants that may be used in conjunction with this property and are defined as follows.

Constant	Value
VSSITEM_PROJECT	0
VSSITEM_FILE	1

If the VSSItem is a project, its Type property will return VSSITEM_PROJECT; if it is a file, its Type property returns VSSITEM_FILE.

The following Visual Basic code uses the Type property to distinguish between file and project objects:

```
Dim objVSSProject As VSSItem
Dim objVSSObject As VSSItem
<...>
' Iterate through all non-deleted items in the project
For Each objVSSObject In objVSSProject.Items(False)
   If objVSSObject.Type = VSSITEM_FILE Then
      Debug.Print ("I am a non-deleted file named: " +  objVSSObject.Name)
   ElseIf objVSSObject.Type = VSSITEM_PROJECT
      Debug.Print ("I am a non-deleted project named: " + _
      objVSSObject.Name)
   End If
Next
```

The Type property applies to both file and project objects.

Version(Version As Variant) As VSSItem (Read Only)

The Version property of the VSSItem is used to reference a specific version within the Version Collection. It accepts a single parameter:

Version: This required Variant parameter is used to identify the specific version you are interested in. The variant version parameter is typically in one of three formats:

- A valid date string. The version is retrieved based on the state of the file or project at that particular date. For example: "6/12/96".
- A valid numerical string. This represents the particular numbered version of the file or project. For example: "12".
- If it is neither a date nor a number it is treated as a label. For example: "Beta1".

The Version collection is one-based. The following Visual Basic code uses the Version property to create an instance of version 3 of the file object:

```
Dim objVSSFile As VSSItem
Dim objVSSVersion3 As VSSItem
Dim VersionNum As Long
VersionNum = 3
<...>
Set objVSSVersion3 = objVSSFile.Version(VersionNum)
Debug.Print Str(objVSSVersion3.VersionNumber)
```

In the preceding sample, objVSSVersion3 now refers to version 3 of the file or project object. You could now call the object's Get or CheckOut methods (if multiple CheckOuts are enabled) to get or check out this version.

The Version property applies to both file and project objects.

VersionNumber As Long (Read-only)

The VersionNumber property of the VSSItem may be used to return the version number of a VSSItem. Each time a new version of a VSSItem is created, SourceSafe increments the version number of the object and applies it to the new version. The following Visual Basic code demonstrates how to retrieve the version number of a VSSItem:

```
Dim objVSSFile As VSSItem
<...>
Debug.Print Str(objVSSFile.VersionNumber)
```

The VersionNumber property applies to both file and project objects.

Versions([iFlags As Long]) As IVSSVersions (Read-only)

The Versions property of a VSSItem is used to reference the Versions collection of the object. This property accepts a single optional parameter:

iFlags: This optional Long parameter is used to set any flags when creating an instance of the Versions collection. The default for this parameter is 0. Valid flags for this parameter include:

VSSFLAG_RECURSNO

VSSFLAG_RECURSYES

VSSFLAG_HISTIGNOREFILES

The following Visual Basic code uses the Versions property to determine which versions in the Versions collection, if any, represent a label:

```
Dim objVSSObject As VSSItem
Dim objVSSVersion As VSSVersion
<...>
For Each objVSSVersion In objVSSObject.Versions

   ' If version represents a label then its 'action' property contains 'label'
   If Left(objVSSVersion.Action, 5) = "Label" Then
      MsgBox("Version " + Str(objVSSVersion.VersionNumber) + _
      " represents a label.")
   End If
Next
```

This next example demonstrates how to create an instance of the Versions collection of a project object that excludes file histories:

```
Dim objVersionsCollection As IVSSVersions
Dim objVSSProject As VSSItem
<...>
Set objVersionsCollection = _
objVSSProject.Versions(VSSFLAG_HISTIGNOREFILES)
```

The Versions property applies to both file and project objects.

Methods

Add(Local As String, [Comment As String], [iFlags As Long])

The Add method of the VSSItem is used to add a file to a project object. This method accepts three parameters:

- **Local:** This required parameter is a string value equal to the complete path of the file you wish to add to the project.
- **Comment:** This optional parameter is a string value set to the comment you want to apply to the Add event. The default for this parameter is an empty string.
- **iFlags:** This optional parameter holds any flag settings you want applied to the Add event. Valid flags include: VSSFLAG_BINBINARY, VSSFLAG_BINTEST, VSSFLAG_BINTEXT, VSSFLAG_DELNO, and VSSFLAG_DELYES.

The following Visual Basic code uses the Add method to add a file to a project:

```
Dim objVSSProject As VSSItem
Dim strLocalFilePath As String
Dim strMyComment As String
strLocalFilePath = "C:\MyFile.txt"
strMyComment = "Add Comment"
<...>
' Add the file
objVSSProject.Add Local:= strLocalFilePath, Comment:= strMyComment,  _
iFlags:=0
```

The Add method applies to project objects only. If you attempt to call the Add method against a file object, the run-time error -2147210887 will be generated. If you attempt to add a file that already exists in the project, the run-time error -2147166572 will be generated.

Branch([Comment As String], [iFlags As Long])

The Branch method of the VSSItem is used to branch a file object that has been shared between one or more projects. This method takes two parameters:

- **Comment:** This optional parameter is a string value set to the comment you want to apply to the Branch event. The default for this parameter is an empty string.
- **iFlags:** This optional parameter holds any flag settings you want applied to the Branch event. When you branch a file for which a working folder has been set, SourceSafe does a silent **Get** to the working folder. The *iFlags* parameter may be used to specify options for that **Get**. See the "VSSFlags" section of this article for more information on valid flags. The default for this parameter is 0.

The following Visual Basic code uses the Branch method to branch a file:

```
Dim objFiletoBranch As VSSItem
Dim strComment As String
strComment = "Branch Comment"
<...>
```

```
objFiletoBranch.Branch Comment:= strComment, iFlags:=0
```

The Branch method applies to file objects only. If you attempt to branch a file that is not shared, the run-time error -2147166405 will be generated. If you attempt to branch a project object, the run-time error -2147210888 will be generated.

CheckIn([Comment As String], [Local As String], [iFlags As Long])

The CheckIn method of the VSSItem is used to check in files that have been checked out. This method accepts three parameters:

- **Comment:** This optional parameter is a string value set to the comment you want to apply to the CheckIn event. The default for this parameter is an empty string.

- **Local:** This optional parameter is a string value set to the complete local path from which you want to check the file in. This path includes the file name that you wish to check in. The default for this parameter is the folder to which the file was checked out. If the file is not found in that location, the operating system's current directory is searched. It is recommended to always specify the Local parameter when calling the **CheckIn** method.

- **iFlags:** This optional parameter holds any flag settings you want applied to the CheckIn event. See the "VSSFlags" section of this article for more information on valid flags. The default for this parameter is 0.

The following Visual Basic code uses the CheckIn method to check in a file:

```
Dim strCheckOutFolder As String
Dim objCheckOut As VSSCheckOut
' Determine where the file is checked out to
For Each objCheckOut In objVSSFile.CheckOuts

' In this sample we will assume that multiple
' checkouts are disabled in Admin so the first
' CheckOut we encounter that is checked out by
' the current user will be the one we are looking for
If objCheckOut.UserName = objVSSDatabase.UserName Then
   strCheckOutFolder = objCheckOut.LocalSpec
   Exit For
End If

Next

' Check in the file
objVSSFile.Checkin Comment:="My CheckIn Comment", _
Local:=strCheckOutFolder + objVSSFile.Name, iFlags:=0
```

The CheckIn method may be called against both file and project objects. When called against a project, any file that can be checked in by the user will be checked in. If you are logged on as user Admin, this includes all checked out files, even those checked out to other users.

CheckOut([Comment As String], [Local As String], [iFlags As Long])

The CheckOut method of the VSSItem is used to check out files from the database. This method accepts three parameters:

- **Comment:** This optional parameter is a string value set to the comment you want to apply to the

CheckOut event. The default for this parameter is an empty string.

- **Local:** This optional parameter is a string value set to the complete local path to which you want to check the file out. This path includes the file name that you wish to check out. The default for this parameter is the **LocalSpec** of the **VSSItem** file item.

- **iFlags:** This optional parameter holds any flag settings you want applied to the CheckOut event. See the "VSSFlags" section of this article for more information on valid flags. The default for this parameter is 0.

The following Visual Basic code uses the CheckOut method to check out a file non-exclusively:

```
Dim strWorkingFolder As String
Dim objVSSFile As VSSItem
<...>
' Retrieve the working folder for the file
strWorkingFolder = objVSSFile.Parent.LocalSpec
If strWorkingFolder = "" Then
   MsgBox("You must set a working folder for this file before it can _
   be checked out." )
Else

   ' Check out the file
   objVSSFile.CheckOut Comment:="", Local:=strWorkingFolder + _
   objVSSFile.Name, iFlags:= VSSFLAG_CHKEXCLUSIVENO
End If
```

The CheckOut method may be called against both file and project objects. When called against a project, any file in the project that can be checked out by the user will be checked out. This means if multiple CheckOuts are enabled for the database and the file is text based, the file will be checked out even if it is checked out by other users.

Destroy()

The Destroy method of the VSSItem purges the item from the database. Whether the file is deleted or non-deleted, calling the Destroy method removes the item from the database entirely. The following Visual Basic code uses the Destroy method to remove a file from the database:

```
Dim objVSSFile As VSSItem
<...>
objVSSFile.Destroy
```

The Destroy method may be called against both file and project objects. Use this method with caution! Once a file or project is destroyed, it cannot be recovered.

Note When calling the Destroy method of a project, all of its subprojects and files are destroyed as well.

Get([Local As String], [iFlags As Long])

The Get method of the VSSItem is used to get files from the database. This method accepts two parameters:

- **Local:** This optional parameter is a string value set to the complete local path to which you want to get the file. This path includes the file name that you wish to get. The default for this parameter is the **LocalSpec** of the **VSSItem** file item.

- **iFlags:** This optional parameter holds any flag settings you want applied to the Get event. See the "VSSFlags" section of this article for more information on valid flags. The default for this parameter is 0.

The following Visual Basic code uses the Get method to get a file. In this example, we set the flags to prevent overwriting a writable copy:

```
Dim strWorkingFolder As String
Dim objVSSFile As VSSItem
<...>
' Retrieve the working folder for the file
strWorkingFolder = objVSSFile.Parent.LocalSpec
If strWorkingFolder = "" Then
   MsgBox("You must set a working folder for this file before it can _
   be gotten." )
Else

   ' Get the file
   objVSSFile.Get Local:= strWorkingFolder + objVSSFile.Name, _
iFlags:= VSSFLAG_REPSKIP
   End If
```

The Get method may be called against both file and project objects. When called against a project, all files in the project are gotten regardless of their CheckOut status.

Label(Label As String, [Comment As String])

The Label method of the VSSItem is used to apply a label to a file or project object. It accepts two parameters:

- **Label:** A required string variable that contains the label to be applied to the object.

- **Comment:** This optional parameter is a string value set to the comment you want to apply to the Label event. The default for this parameter is an empty string.

The following Visual Basic code uses the Label method to label a file:

```
Dim objVSSFile As VSSItem
Dim strLabelText As String
Dim strLabelCommentText As String
<...>
objVSSFile.Label Label:= strLabelText, Comment:= strLabelCommentText
```

Note Labels applied to project objects will be inherited by all subprojects and files within the project. If you apply a label that already exists, the original label is removed.

Move(NewParent As VSSItem)

The Move method of the VSSItem is used to move a VSSItem from one project to another. The Move method accepts one parameter:

NewParent: This required *VSSItem* parameter represents the project to which the object is to be moved.

The following Visual Basic code uses the Move method to move a project:

```
Dim objVSSProjectSource As VSSItem
Dim objVSSProjectTarget As VSSItem
<...>
objVSSProjectSource.Move objVSSProjectTarget
MsgBox("Project " + objVSSProjectSource.Name + " has been moved to " + _
objVSSProjectTarget.Name)
```

The Move method applies to project objects only. If you attempt to access the Move property of a file object, the run-time error -2147210887 will be generated. If you attempt to move a project to its current parent, the run-time error -2147166574 will be generated.

Note Use this method with caution! Once a project is moved, attempting to get a previous version of the original project will no longer be successful.

NewSubproject(Name As String, [Comment As String])

The NewSubproject method of the VSSItem is used to create new projects under the current project object. The NewSubproject method accepts two parameters:

- **Name:** This required string parameter contains the name of the new project to create. This new project will be created under the project object from which the **NewSubproject** method was called.
- **Comment:** This optional parameter is a string value set to the comment you want to apply to the NewSubproject event. The default for this parameter is an empty string.

The following Visual Basic code uses the NewSubproject method to add a new project to the database and then create an instance of the new project:

```
Dim objVSSProject As VSSItem
Dim strNewProjectName As String
Dim strNewProjectComment As String
<...>
objVSSProject. NewSubproject Name:= strNewProjectName, Comment:= _
strNewProjectComment
Set objNewProject = objVSSDatabase.VSSItem(objVSSProject.Spec ,  False)
```

The NewSubproject method applies to project objects only. If you attempt to access the NewSubproject property of a file object, the run-time error -2147210887 will be generated. If you attempt to add a project that already exists, the run-time error -2147166572 will be generated.

Share(pIItem As VSSItem, [Comment As String], [iFlags As Long])

The Share method of the VSSItem is used to share a file or project from one project to another. When a file is shared between one or more projects, any change to the file is propagated throughout all projects in which the file is shared. The Share method accepts three parameters:

- **pIItem:** This required parameter is a **VSSItem** that represents the file or project object to be shared.

- **Comment:** This optional parameter is a string value set to the comment you want to apply to the Share event. The default for this parameter is an empty string.

- **iFlags:** This optional parameter holds any flag settings you want applied to the Share event. See the "VSSFlags" section of this article for more information on valid flags. The default for this parameter is 0.

The following Visual Basic code uses the Share method to share a file to an existing project:

```
Dim objVSSProject As VSSItem
Dim objVSSFile As VSSItem
Dim strShareComment As String
<...>
objVSSProject.Share pIItem:=objVSSFile, Comment:= _
strShareComment, iFlags:=0
```

In this example, the file object objVSSFile is being shared to the project object objVSSProject. The Share method may be called against either a file or a project object. When sharing a project, all files within the project are shared (but the project object is not shared). If you use the flag VSSFLAG_RECURSYES, all files in subprojects are shared as well.

In Visual Basic, if you attempt to share a file (for example, File.txt) into a project that already contains a file of the same name (File.txt), the run-time error -2147166572 will be generated. If you attempt to recursively share a project to one of its descendants, run-time error -2147166417 will be generated.

UndoCheckOut([Local As String], [iFlags As Long])

The UndoCheckOut method of the VSSItem is used to undo checkout of a checked-out file. The UndoCheckOut method accepts two parameters:

- **Local:** This optional parameter is a string that contains the complete path and file name from which you want to undo the checkout. The default for this parameter is the folder to which the file was checked out. If the file is not found in that location, the operating system's current directory is searched. It is recommended that you always specify the Local parameter when calling the **UndoCheckOut** method.

- **iFlags:** This optional parameter holds any flag settings you want applied to the UndoCheckOut event. See the "VSSFlags" section of this article for more information on valid flags. The default for this parameter is 0.

The following Visual Basic code uses the UndoCheckOut method to uncheck out a file from its working folder:

```
    Dim objVSSFile As VSSItem
    Dim strCheckOutFolder As String
    <...>
    strCheckOutFolder = objVSSCheckOut.LocalSpec
    objVSSFile.UndoCheckOut Local:= strCheckOutFolder + objVSSFile.Name, _
    iFlags:=0
```

This sample assumes the file was checked out to the working folder. The UndoCheckOut method may be used against both file and project objects. When called against a project, any file that can be unchecked out by the user will be unchecked out. If you are logged on as user Admin, this includes all checked out files, even those checked out to other users.

VSSItemType Constants

The following predefined constants may be used in conjunction with the VSSItemType property to distinguish between files and projects:

- **Const VSSITEM_FILE**
- If the **VSSItemType** property returns this value, the object represents a file. The value of this constant is 1.
- **Const VSSITEM_PROJECT**
- If the **VSSItemType** property returns this value, the object represents a project.

  The value of this constant is 0.

The following Visual Basic code demonstrates the use of these constants:

```
Dim objVSSProject As VSSItem
Dim objVSSObject As VSSItem
<...>
' Iterate through all non-deleted items in the project
For Each objVSSObject In objVSSProject.Items(False)
   If objVSSObject.Type = VSSITEM_FILE Then
      Debug.Print ("I am a non-deleted file named: " +  objVSSObject.Name)
   ElseIf objVSSObject.Type = VSSITEM_PROJECT
      Debug.Print ("I am a non-deleted project named: " + _
      objVSSObject.Name)
   End If
Next
```

VSSRights Constants

The following six predefined constants may be used in conjunction with the VSSDatabase object's DefaultProjectRights property and the VSSUser's ProjectRights property.

These constants have the following values.

Constant	Value
VSSRIGHTS_READ	1
VSSRIGHTS_CHKUPD	2
VSSRIGHTS_ADDRENREM	4
VSSRIGHTS_DESTROY	8
VSSRIGHTS_ALL	15
VSSRIGHTS_INHERITED	16

DefaultProjectRights: The SourceSafe Administrator has the ability to add new users to the SourceSafe database Users collection. In Automation, this is accomplished through the AddUser method of the database object. The DefaultProjectRights property is used to set or return the default rights for a new user added to the SourceSafe database. The only valid settings for DefaultProjectRights property are as follows.

Value	**Rights**
0	No rights
1	Read rights
3	Read /check out/check in
7	Read /check out/check in/add/rename rights
15	Read /check out/check in/add/rename/destroy rights

The following Visual Basic code uses the VSSRights constants to set the DefaultProjectRights property of a database object:

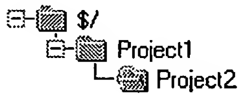
```
Dim DefaultRights As Long
<...>
   DefaultRights = VSSRIGHTS_ADDRENREM + VSSRIGHTS_CHKUPD + _
   VSSRIGHTS_READ
   ' Set default rights
      objVSSDatabase.DefaultProjectRights = DefaultRights
```

Only user Admin may set the DefaultProjectRights property. If any other user attempts to set the DefaultProjectRights property of the database, the run-time error -2147166522 is generated.

See the "DefaultProjectRights Property" topic of the "VSSDatabase Object" section for more information.

ProjectRights: Each VSSUser object has a ProjectRights property that defines the rights the user has to a specific project. For these rights to be operative, the ProjectRightsEnabled property of the database object must be set to true (this is the equivalent to selecting the Enable project security check box in the Admin utility).

The user Admin may set these rights for any user on a project by project basis. If a user's rights have been set for a specific project, and the project's subprojects (if any) have not been set, the rights are inherited down the project path. See, for example, the following project structure:



If user Admin gives a user read-only rights to $/Project1 and does not set any rights for the user to $/Project1/Project2, the user will automatically have read-only rights to $/Project1/Project2 as well. This behavior is identical to that of the SourceSafe Explorer and Admin utility.

User Admin may use the ProjectRights property to set or retrieve the current project rights for a user (see the "ProjectRights Property" topic of the "VSSUser Object" section for more information). If the user's rights for the current project are inherited from the parent project, the ProjectRights property for the user will include the constant VSSRIGHTS_INHERITED. The developer may use this to distinguish between projects whose rights have been explicitly set as opposed to those that have been inherited.

The following Visual Basic code uses the VSSRights constants to set and retrieve the ProjectRights property of a VSSUser object:

```
Dim CurrentUser As VSSUser
Dim strUserName As String
Dim str ProjectName As String
' Set the variables
strUserName ="Guest"
str ProjectName = "$/"
<...>
' Create an instance of the VSSUser Object
Set CurrentUser = objVSSDatabase.User(strUserName)
' Retrieve the rights for project
Select Case CurrentUser.ProjectRights(strProjectName)

        Case VSSRIGHTS_ALL

      MsgBox("User has destroy rights to project " + ProjectName + ".")
   Case VSSRIGHTS_INHERITED + VSSRIGHTS_ALL

      MsgBox("User has inherited destroy rights to project " + ProjectName + "

        Case VSSRIGHTS_ADDRENREM + VSSRIGHTS_CHKUPD + _
      VSSRIGHTS_READ
      MsgBox("User has add/rename/delete rights to project " + _
      ProjectName + ".")
   Case VSSRIGHTS_ADDRENREM + VSSRIGHTS_CHKUPD + _
      VSSRIGHTS_READ + VSSRIGHTS_INHERITED

      MsgBox("User has inherited add/rename/delete rights to project " + _
      ProjectName + ".")

   Case VSSRIGHTS_CHKUPD + VSSRIGHTS_READ
      MsgBox("User has check out/check in rights to project " + ProjectName +
   Case VSSRIGHTS_CHKUPD + VSSRIGHTS_READ + _
      VSSRIGHTS_INHERITED

      MsgBox("User has inherited check out/check in rights to project " + _
      ProjectName + ".")

        Case VSSRIGHTS_READ
      MsgBox("User has read-only rights to project " + ProjectName + ".")
   Case VSSRIGHTS_READ + VSSRIGHTS_INHERITED

      MsgBox("User has inherited read-only rights to project " + _
      ProjectName + ".")

     End Select
' Set the rights for the current user and project
CurrentUser.ProjectRights(CurrentProject) = VSSRIGHTS_ALL
```

Although the developer can set the ProjectRights property to a value that includes the constant VSSRIGHTS_INHERITED, it is not supported. For example:

```
CurrentUser.ProjectRights(CurrentProject) = VSSRIGHTS_ALL + _
VSSRIGHTS_INHERITED
```

is incorrect syntax. The Automation interface will automatically set the ProjectRights property to include VSSRIGHTS_INHERITED when appropriate. Although no run-time error is generated when the ProjectRights property is programmatically set to include VSSRIGHTS_INHERITED, the results may be unexpected.

Only user Admin may set the ProjectRights property of a VSSUser object. If any other user attempts to set the ProjectRights property of a user, the run-time error -2147166522 is generated.

See the "ProjectRights Property" topic of the "VSSDatabase Object" section for more information.

VSSUser Object

The VSSUser object represents a user in the current SourceSafe database object. This section describes the properties and methods of the VSSUser object.

Properties

Name As String

The Name property of the VSSUser is used to return or set the name of the current SourceSafe user. The following Visual Basic code demonstrates retrieving and setting the Name property:

```
Dim strUserName As String
Dim objCurrentUser As VSSUser
Dim strNewName As String
' Set the variables
strUserName ="Guest"
<...>
'Create an instance of the VSSUser Object
Set objCurrentUser = objVSSDatabase.User(strUserName)
' Display the name property
MsgBox(objCurrentUser.Name)
' Rename the User
objCurrentUser.Name = strNewName
```

Any user may retrieve the Name property of the VSSUser, however only user Admin can set the Name property of a user. The Name property for user Admin may not be changed. If you attempt to change the Name property for user Admin, the run-time error -2147164975 will be generated. If a user other than Admin attempts to set the Name property, the run-time error -2147166522 is generated.

Password As String

The Password property of the VSSUser is used to return or set the password of the current SourceSafe user. The following Visual Basic code demonstrates retrieving and setting the Password property:

```
Dim strUserName As String
Dim objCurrentUser As VSSUser
Dim strNewPassword As String
' Set the variables
strUserName ="Guest"
```

```
<...>
' Create an instance of the VSSUser Object
Set objCurrentUser = objVSSDatabase.User(strUserName)
 ' Display the password property
MsgBox(objCurrentUser.Password)
' Set the password property
objCurrentUser. Password = strNewPassword
```

Only user Admin and the current user may access the Password property of the current user. In other words, user Guest may retrieve and set their own password but not that of others. User Admin may retrieve and set any user's password. If any other attempts to set the Password property of another user, the run-time error -2147166522 is generated.

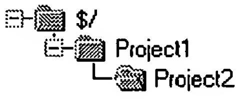
ProjectRights([Project As String = "$/"]) As Long

Each VSSUser object has a ProjectRights property that defines the rights the user has to a specific project. For these rights to be operative, the ProjectRightsEnabled property of the database object must be set to true. (This is the equivalent to selecting the Enable project security check box in the Admin utility)

The ProjectRights property accepts a single parameter:

Project: This optional string variable contains the SourceSafe project path whose ProjectRights properties you are interested in. The default for this parameter is $/ (the root project).

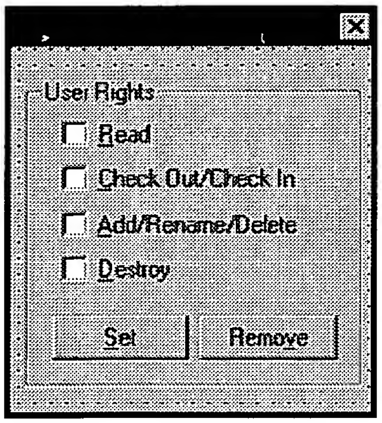
The user Admin may set or retrieve the ProjectRights for any user on a project by project basis. If a user's rights have been set for a specific project and the project's subprojects (if any) have not been set, the rights are inherited down the project path. See, for example, the following project structure:



If user Admin gives a user read-only rights to $/Project1 and does not set any rights for the user to $/Project1/Project2, the user will automatically have read-only rights to $/Project1/Project2 as well. This behavior is identical to that of the SourceSafe Explorer and Admin utility.

If a user's rights for the current project are inherited from the parent project, the ProjectRights property for the user will include the constant VSSRIGHTS_INHERITED. The developer may use this to distinguish between projects whose rights have been explicitly set as opposed to those that have been inherited.

The following Visual Basic code sets the current user's project rights based on a series of check boxes named chkRead, chkUpdate, chkAddRenameDelete, and chkDestroy. This sample code could be associated with a form object that looks like this.



The following code would appear in the Click event of the Command button captioned "Set":

```
   Dim objCurrentUser As VSSUser
   Dim strCurrentProject As String
   Dim strUserName As String
' Set the variables
strUserName ="Guest"
strCurrentProject = "$/"
' Create an instance of the VSSUser Object
   Set objCurrentUser = objVSSDatabase.User(strUserName)
   <...>
   ' Checkbox captioned 'Read'
If chkRead.Value = 0 Then
          objCurrentUser.ProjectRights(strCurrentProject) = 0

   ' Check boxes captioned 'Read' and 'CheckOut/CheckIn
      ElseIf chkRead.Value = 1 And chkUpdate.Value = 0 Then
          objCurrentUser.ProjectRights(strCurrentProject) = VSSRIGHTS_READ

   ' Check boxes captioned 'CheckOut/CheckIn and Add/Rename/Delete
   ElseIf chkUpdate.Value = 1 And chkAddRenameDelete.Value = 0 Then
   objCurrentUser.ProjectRights(strCurrentProject) = _
   VSSRIGHTS_CHKUPD + VSSRIGHTS_READ

' Check boxes captioned Add/Rename/Delete and Destroy
      ElseIf chkAddRenameDelete.Value = 1 And chkDestroy.Value = 0 Then
          objCurrentUser.ProjectRights(strCurrentProject) =
     VSSRIGHTS_ADDRENREM + VSSRIGHTS_CHKUPD +  _
     VSSRIGHTS_READ

   ' Check box captioned 'Destroy'
      ElseIf chkDestroy.Value = 1 Then
          objCurrentUser.ProjectRights(strCurrentProject) = VSSRIGHTS_ALL
      End If
```

Only user Admin may set the ProjectRights property of a user. The ProjectRights property for user Admin may not be changed. If you attempt to change the ProjectRights property for user Admin, the run-time error -2147164975 will be generated. If any user other than Admin attempts to access the ProjectRights property of a user, the run-time error -2147166522 is generated.

ReadOnly As Boolean

Visual SourceSafe provides two levels of security: the security provided by the user's ProjectRights property and the security provided by the ReadOnly property. When the

ReadOnly property is set to true, the user will have read-only rights to the database regardless of any other rights settings. In other words, the VSSUser.ReadOnly property overrides the VSSUser.ProjectRights property of the user.

The following Visual Basic code sets the ReadOnly property of user Guest to true:

```
Dim strUserName As String
Dim objCurrentUser As VSSUser
' Set the variables
strUserName ="Guest"
<...>
' Create an instance of the VSSUser Object
Set objCurrentUser = objVSSDatabase.User(strUserName)
objCurrentUser.ReadOnly = True
```

Only user Admin may set the ReadOnly property of a user. The ReadOnly property for user Admin may not be changed. If you attempt to change the ReadOnly property for user Admin, the run-time error -2147164975 will be generated. If any user other than Admin attempts to access the ReadOnly property of a user, the run-time error -2147166522 will be generated.

Methods

Delete()

The Delete method of the VSSUser object is used to delete the user from the current SourceSafe database. The following Visual Basic code sample deletes user Guest from the current database object:

```
Dim objUserToDelete As VSSUser
Dim strUserName As String
' Set the variables
strUserName ="Guest"
<...>
' Create an instance of the user object
Set objUserToDelete = objVSSDatabase.User(strUserName)
' Delete user
 objUserToDelete.Delete
```

Only user Admin may call the Delete method of a user. The Delete method for user Admin may not be called. If you attempt to call the Delete method for user Admin, the run-time error -2147166521 will be generated. If any user other than Admin attempts to call the Delete method, the run-time error -2147166522 will be generated.

RemoveProjectRights(Project As String)

The RemoveProjectRights method of the VSSUser object is used to remove a user's project rights from a specific project. The RemoveProjectRights method accepts a single parameter:

Project: This required string variable contains the complete project path of the project you want to manipulate.

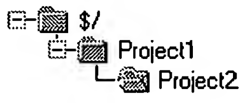
The following Visual Basic code sample uses the RemoveProjectRights method to remove user Guest's project rights from $/ (the root project):

```
    Dim objCurrentUser As VSSUser
    Dim strUserName As String
    Dim strCurrentProject As String
```

```
' Set the variables
   strUserName ="Guest"
   strCurrentProject = "$/"
' Create an instance of the user object
   Set objCurrentUser = objVSSDatabase.User(strUserName)

   ' Remove the user's project rights
   objCurrentUser.RemoveProjectRights (strCurrentProject)
```

When a user's project rights are removed from a project, the project will inherit its rights from its parent project. For example, if we have the following project tree:



where no explicit project rights are set for $/, rights for $/Project1have been set to read-only and rights for $/Project1/Project2 have been set to destroy. If we use the RemoveProjectRights method to remove the user's rights for $/Project1/Project2, this project will now inherit its rights from $/Project1 and will consequently give the user read-only rights to the project.

Only user Admin may call the RemoveProjectRights method of a user. The RemoveProjectRights property for user Admin may not be called. If you attempt to call the RemoveProjectRights method for user Admin, the run-time error -2147166521 will be generated. If any user other than Admin attempts to call the RemoveProjectRights method, the run-time error -2147166522 will be generated.

VSSVersion Object

The VSSVersion object represents a version of a file or project item. This object may represent the current or any previous version available in the item's history. The VSSVersion object has the following properties.

Properties

Action As String (Read-only)

The Action property of the VSSVersion object is a string variable that returns the action that created the version. For example, if a file is checked in, the version created will have an action of CheckedIn*<SourceSafe file path>.* Possible values for the Action property include:

- Added
- Branch at version <version number>
- Checked in <SourceSafe file path>
- Created
- Deleted
- Destroyed

- Labeled <label>
- Moved <old SourceSafe path> to <new SourceSafe path>
- Renamed <old name> to <new name>
- Shared <SourceSafe file path>

The following Visual Basic code iterates through the Versions collection and identifies the action that created the version:

```
Dim objVSSObject As VSSItem
Dim objVSSVersion As VSSVersion
<...>
For Each objVSSVersion In objVSSObject.Versions

   ' Display the action
   MsgBox("Version " + Str(objVSSVersion.VersionNumber) + _
   " has an action of " + objVSSVersion.Action  + ".")
Next
```

Comment As String (Read-only)

The Comment property of the VSSVersion object is a string variable that returns the comment (if any) applied to the version by the user. The following Visual Basic code iterates through the Versions collection of a file and identifies those that have had a comment applied to them:

```
Dim objVSSObject As VSSItem
Dim objVSSVersion As VSSVersion
<...>
For Each objVSSVersion In objVSSObject.Versions

   ' If version has a comment
   If objVSSVersion.Comment <> "" Then
      MsgBox("Version " + Str(objVSSVersion.VersionNumber) + _
      " has a comment of  " + objVSSVersion.Comment + ".")
   End If
Next
```

Date As Date (Read-only)

The Date property of the VSSVersion object is a date variable that returns the date the version was created. The following Visual Basic code iterates through the Versions collection of a file and displays the date of each version:

```
Dim objVSSObject As VSSItem
Dim objVSSVersion As VSSVersion
<...>
For Each objVSSVersion In objVSSObject.Versions
   MsgBox("Version " + Str(objVSSVersion.VersionNumber) + _
   " was created  " + Str(objVSSVersion.Date) + ".")
   Next
```

Label As String (Read-only)

The Label property of the VSSVersion object is a string variable that returns the label applied to the version by the user. The following Visual Basic code iterates through the Versions collection of a file and identifies those that have had a label applied to them:

```
Dim objVSSObject As VSSItem
Dim objVSSVersion As VSSVersion
<...>
For Each objVSSVersion In objVSSObject.Versions

   ' If version represents a Label then its 'action' property contains 'label'
   If Left(objVSSVersion.Action, 5) = "Label" Then
      MsgBox("Version " + Str(objVSSVersion.VersionNumber) + _
      " has been labeled " + objVSSVersion.Label + ".")
   End If
```

LabelComment As String (Read-only)

The LabelComment property of the VSSVersion object is a string variable that returns the comment applied to a label by the user. The following Visual Basic code iterates through the Versions collection of a file and identifies those that have had a label applied to them and displays the label comment (if any):

```
Dim objVSSObject As VSSItem
Dim objVSSVersion As VSSVersion
<...>
For Each objVSSVersion In objVSSObject.Versions

   ' If version represents a Label then its 'action' property contains 'label'
   If Left(objVSSVersion.Action, 5) = "Label" Then
      If objVSSVersion.LabelComment <> "" Then
         MsgBox("Version " + Str(objVSSVersion.VersionNumber) + _
         " has a label comment of " + objVSSVersion.LabelComment + ".")
         EndIf
   End If
```

Next

Only versions that contain a label will contain a label comment.

UserName As String (Read-only)

The UserName property of the VSSVersion object is a string variable that returns the UserName that created the version. The following Visual Basic code iterates through the Versions collection of a file and displays the UserName for each version:

```
Dim objVSSObject As VSSItem
Dim objVSSVersion As VSSVersion
<...>
For Each objVSSVersion In objVSSObject.Versions
   MsgBox("Version " + Str(objVSSVersion.VersionNumber) + _
   " was created  by " + objVSSVersion.UserName + ".")
```

VersionNumber As String (Read-only)

The VersionNumber property of the VSSVersion object is a string variable that returns the version number of the version. Version numbers are one-based and grow larger by increments of one as each new version is added. The following Visual Basic code iterates through the Versions collection of a file and displays the version number of each version:

```
Dim objVSSObject As VSSItem
Dim objVSSVersion As VSSVersion
<...>
For Each objVSSVersion In objVSSObject.Versions
   MsgBox("This is version number" + _
   Str(objVSSVersion.VersionNumber)
```

VSSItem As VSSItem (Read-only)

The VSSItem property of the VSSVersion object allows you to manipulate a specific version in an item's history. For example, if you have a file item that has 50 versions, you could get version 12 by issuing the Get method against the version. The following Visual Basic code demonstrates getting a previous version of a file:

```
Dim objVSSVersion  As VSSItem
Dim ObjVSSObject As VSSItem
Dim VersionNumber As Integer
Dim GetPath As String
Dim GetFlags As Long
<...>
' Assume the file has more than 12 versions
VersionNumber = 12
Set objVSSVersion = _
objVSSObject.Version(VersionNumber)
objVSSVersion.Get Local:=GetPath, iFlags:=GetFlags
```

Error Messages

The following list describes the more common run-time errors encountered in Automation. The symbol < > represents a parameter that will be inserted into the error message.

Error number	Error description
-2147164975	You cannot change the Admin name or access privileges.
-2147166139	Invalid DOS path: < >
-2147166373	File < > is currently checked out by < >.
-2147166374	You currently have file < > checked out.
-2147166386	Cannot delete the root project.
-2147166391	You do not have access rights to < >.
-2147166398	Cannot move the root project.
-2147166404	File < > is not checked out.
-2147166405	File < > is not shared by any other projects.
-2147166411	Cannot Rename the root project.
-2147166417	A project cannot be shared under a descendant.
-2147166418	File < > is already shared by this project.
-2147166424	The SourceSafe database has been locked by the Administrator.
-2147166519	Invalid password.
-2147166521	Cannot delete the Admin user.
-2147166522	Permission denied.
-2147166525	User "< >" already exists.
-2147166526	User "< >" not found.
-2147166572	An item with the name < > already exists.
-2147166574	You can not move a project under itself.
-2147166583	Version not found.
-2147167977	The SourceSafe database path < > does not exist. Please select another database.
-2147210887	This command only works on projects.
-2147210888	This command only works on files.
-2147352566	Invalid access code (bad parameter)*.

*Many SourceSafe parameters have a maximum limit. For example, the name of a SourceSafe user may not exceed 31 characters. When limits such as this are exceeded, the run-time error -2147352566 is typically returned. For detailed information on limits imposed by SourceSafe, please see the Knowledge Base article Q138298, titled "Visual SourceSafe System Capacities and Specifications."

Putting It All Together—Driving a SourceSafe Database

Setting Up the Visual Basic Development Environment

If you want to create an application that uses the SourceSafe Automation interface, you need to set a reference to the SourceSafe 6.0 Type Library. If this option doesn't appear in your Visual Basic Reference list, use the Browse command from the References dialog box to locate the file ssapi.dll. By default, this file will reside in the WIN32 folder of your Visual SourceSafe 6.0 installation.

If you want to use the ActiveX Diff-Merge Control in your project, you need to register the file DiffMergeCTL.ocx (using RegSvr32.exe), and then set a reference to the *DiffMergeCTL ActiveX Control Module*. The file DiffMergeCTL.ocx must be placed in the same folder as the file ssus.dll (...\VSS\WIN32 by default). You can download the ActiveX Diff-Merge Control from the Visual SourceSafe Web site at http://msdn.microsoft.com/ssafe/. Please note that the ActiveX Diff-Merge Control is not supported.

Writing the Code

This section takes you through a step-by-step process of writing a basic code outline that drives a SourceSafe database. You may copy this code sample and paste it into a Visual Basic 5.0 project to view the sample in action.

One of the first things you need to do when creating an application that uses Automation to drive a SourceSafe database is open the database object. Create a BAS module that declares some of your objects globally, such as:

```
' Global reference to the database
Global objVSSDatabase As New VSSDatabase
'Global reference to the root project
Global objVSSRoot As VSSItem
' Set a reference to the test project
Global objNewProject As VSSItem
' Set a reference to the test file
Global objVSSFile As VSSItem
```

Once the BAS module is created, we can open a new form and add the following code to its Load event:

```
Private Sub Form_Load()
' Declare the Username, password and SourceSafe path variables
Dim UserName As String
Dim SrcSafeIni As String
Dim Password As String
' Set on error routine
On Error Resume Next
' Set the values (customize these as needed)
UserName = "Guest"
SrcSafeIni = "C:\VSS\SRCSAFE.INI"
Password = ""
' Create an instance of the database
objVSSDatabase.Open SrcSafeIni, UserName, Password
```

```
    ' Check for errors
    Select Case Err

    Case 0
              MsgBox("Database opened!")
          ' Create reference to the root
       Set objVSSRoot = objVSSDatabase.VSSItem("$/", False)
       Case Else
       MsgBox("Error logging into SourceSafe!" + vbCrLf + _
       Err.Description)
          End
    End Select
End Sub
```

Now the database has been opened, we can begin to manipulate its contents. First, let's add a project and then add a file to that project. We can put this code into the Click event of a Command button on the form with a caption of "Create Project and Add File."

```
Private Sub Command1_Click()
Dim AddFlags As Long
' Create the project and create an instance of it
objVSSRoot.NewSubproject Name:="MyNewProject", Comment:= _
"My Create Comment"
Set objNewProject = objVSSDatabase.VSSItem("$/MyNewProject",  False)
' Add the file, removing the local copy
AddFlags = VSSFLAG_DELYES
objNewProject.Add Local:="C:\Test.txt", Comment:="My Add Comment",  _
iFlags:=AddFlags
' Create an instance of a reference to the new file
Set objVSSFile = objVSSDatabase.VSSItem(objNewProject.Spec + _
"/Test.txt", False)
End Sub
```

Now that we have added a file to the database, we can set its working folder and then check it out. To do this, add the following code to another Command button on the form with a caption of "Set Working Folder and CheckOut":

```
Private Sub Command2_Click()
' Set the Working Folder
objNewProject.LocalSpec = "C:\"

' Check out the file
objVSSFile.CheckOut Comment:="My CheckOut Comment", _
Local:=objNewProject.LocalSpec + objVSSFile.Name, iFlags:=0
End Sub
```

Note   We could have used the *iFlags* parameter for a variety of things here. For example, VSSFLAG_REPSKIP and VSSFLAG_TIMENOW would set the TimeStamp of the local file and provide a local writable copy that didn't exist. See the "VSSFlags" section for more information.

Now that we have checked a file out, and assuming the local copy has been modified, we can check it back in. Place the following code into the Click event of another Command button with a caption of "Check In":

```
Private Sub Command3_Click()
Dim CheckOutFolder As String
Dim objCheckOut As VSSCheckOut
' Determine where the file is checked out to
For Each objCheckOut In objVSSFile.CheckOuts
```

```
' In this sample we will assume that multiple CheckOuts
' are disabled in Admin so the first CheckOut we encounter
' that is checked out by the current user will be the one
' we are looking for
If objCheckOut.UserName = objVSSDatabase.UserName Then
CheckOutFolder = objCheckOut.LocalSpec
Exit For
End If

Next

' Check in the file
objVSSFile.CheckIn Comment:="My CheckIn Comment", _
Local:=CheckOutFolder + objVSSFile.Name, iFlags:=0
End Sub
```

Note We could have used the *iFlags* parameter for a variety of things here, such as removing the local copy or keeping the file checked out. See the "VSSFlags" section for more information.

We have covered the fundamental objects, properties, and methods to begin creating your own Visual Basic application to drive a SourceSafe database. For a complete sample, see the Visual SourceSafe Web site at http://msdn.microsoft.com/ssafe/.

Send feedback on this article. Find support options.